Dynamic Knowledge Inference and Learning under Adaptive Fuzzy Petri Net Framework

Xiaoou Li, Wen Yu, *Member, IEEE,* and Felipe Lara-Rosano, *Associate Member, IEEE*

***Abstract*—Since knowledge in expert system is vague and modified frequently, expert systems are fuzzy and dynamic systems. It is very important to design a dynamic knowledge inference framework which is adjustable according to knowledge variation as human cognition and thinking. Aiming at this object, a generalized fuzzy Petri net model is proposed in this paper, it is called adaptive fuzzy Petri net (AFPN). AFPN not only takes the descriptive advantages of fuzzy Petri net, but also has learning ability like neural network. Just as other fuzzy Petri net (FPN) models, AFPN can be used for knowledge representation and reasoning, but AFPN has one important advantage: it is suitable for dynamic knowledge, i.e., the weights of AFPN are ajustable. Based on AFPN transition firing rule, a modified back propagation learning algorithm is developed to assure the convergence of the weights.**

***Index Terms*—Expert system, fuzzy reasoning, knowledge learning, neural network, Petri net.**

I. Introduction

PETRI NETS (PNs) have ability to represent and analyze in an easy way concurrency 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 theory, neural networks, etc. These modified PNs are widely used in computer, manufacturing, robotic, knowledge based systems, process control, as well as other kinds of engineering applications.

PNs have an inherent quality in representing logic in intuitive and visual way, and FPNs take all the advantages of PNs. So, the reasoning path of expert systems can be reduced to simple sprouting trees if FPN-based reasoning algorithms are applied as an inference engine. FPN are also used for fuzzy knowledge representation and reasoning, many results prove that FPN is suitable to represent and reason misty logic implication relations [2], [3], [1], [12], [4], [8].

Knowledge in expert systems is updated or modified frequently, expert systems may be regarded as dynamic systems. Suitable models for them should be adaptable. In other words, the models must have ability to adjust themselves according to the systems' changes. However, the lack of adjustment (learning) mechanism in FPNs can not cope with potential changes of actual systems [5].

Manuscript received June 19, 1999; revised September 1, 2000.

X. Li and W. Yu are with the Sección de Computación, Departamento de Ingeniería Eléctrica, CINVESTAV-IPN, México City, D.F. 07360, México (e-mail: lixo@cs.cinvestav.mx).

F. Lara-Rosano is with the Centro de Instrumentos, Universidad Nacional Autonoma de México (UNAM), A.P. 70-186, Cd. Universitaria, México City, D.F. 04510, México.

Publisher Item Identifier S 1094-6977(00)11205-2.

Recently, some adjustable FPNs were proposed. [3] gave an algorithm to adjust thresholds of FPN, but weights' adjustments were realized by test. [6] proposed a generalized FPN model (GFPN) which can be transformed into neural networks with OR/AND logic neurons [5], thus, parameters of the corresponding neural networks can be learned (trained). In fact, the knowledge learning in [6] was under the framework of neural networks. Adaptive Fuzzy Petri Net (AFPN) [13] has also the learning ability of a neural network, but it does not need to be transformed into neural networks. However the learning algorithm in [13] is based on a special transition firing rule, it is necessary to know certainty factors of each consequence proposition in the system. Obviously, this restriction is too strict for an expert system.

In this paper, we propose a more generalized reasoning rule for AFPN. Back propagation algorithm is developed for the knowledge learning under generalized conditions. The structure of the paper is organized as follows: after the introduction of the FPN and AFPN models, the reasoning algorithm and the weight learning algorithm are developed, examples are included as an illustration.

II. Knowledge Representation and Fuzzy Petri Net

In this section, we will review weighted fuzzy production rules and FPN.

A. Weighted Fuzzy Production Rules

In many situations, it may be difficult to capture data in a precise form. In order to properly represent real world knowledge, fuzzy production rules have been used for knowledge representation [2]. A fuzzy production rule (FPR) is a rule which describes the fuzzy relation between two propositions. If the antecedent portion of a fuzzy production rule contains "AND" or "OR" connectors, then it is called a composite fuzzy production rule. If the relative degree of importance of each proposition in the antecedent contributing to the consequent is considered, Weighted Fuzzy Production Rule (WFPR) has to be introduced [7].

Let R be a set of weighted fuzzy production rules $R = \{R_1, R_2, \ldots R_n\}$. The general formulation of the i_th weighted fuzzy production rule is as follows:

$$R_i : \text{IF } a \text{ THEN } c\,(CF = \mu),\ Th, w$$

where

$a = \langle a_1, a_2, \ldots, a_n \rangle$ antecedent portion which comprises of one or more propositions connected by either "AND" or "OR";

c	consequent proposition;
μ	certainty factor of the rule;
Th	threshold;
w	weight.

In general, WFPRs are categorized into three types which are defined as follows.

Type 1: A Simple Fuzzy Production Rule

$$R: \text{IF } a \text{ THEN } c\,(CF=\mu), \lambda, w$$

For this type of rule, since there is only one proposition a in the antecedent, the weight w is meaningless.

Type 2: A Composite Conjunctive Rule

$$R: \text{IF } a_1 \text{ AND } a_2 \text{ AND } \cdots \text{ AND } a_n \text{ THEN } c\,(CF=\mu), \lambda_1, \lambda_2, \ldots, \lambda_n, w_1, w_2, \ldots, w_n$$

Type 3: A Composite Disjunctive Rule

$$R: \text{IF } a_1 \text{ OR } a_2 \text{ OR } \cdots \text{ OR } a_n \text{ THEN } c\,(CF=\mu), \lambda_1, \lambda_2, \ldots, \lambda_n, w_1, w_2, \ldots, w_n$$

For Type 2 and Type 3, a_i is the ith antecedent proposition of rule R, and c the consequent one. Each proposition a_i can have the format "x is f_a", where f_a is an element of a set of fuzzy sets F. μ are the threshold and certainty factor of a simple or composite rule; λ_i, w_i are the threshold and weight of the ith antecedent of a composite conjuctive or disjunctive rule. In above definition, thresholds are assigned to antecedent propositions. For composite conjuctive rules, thresholds are assigned to the weighted sum of all antecedent propositions.

In this paper, in order to cope with Adaptive Fuzzy Petri Net (AFPN), we define WFPRs as following new forms:

Type 1: A Simple Fuzzy Production Rule

$$R: \text{IF } a \text{ THEN } c\,(CF=\mu), \lambda, w$$

Type 2: A Composite Conjunctive Rule

$$R: \text{IF } a_1 \text{ AND } a_2 \text{ AND } \cdots \text{ AND } a_n \text{ THEN } c\,(CF=\mu), \lambda, w_1, w_2, \ldots, w_n$$

Type 3: A Composite Disjunctive Rule

$$R: \text{IF } a_1 \text{ OR } a_2 \text{ OR } \cdots \text{ OR } a_n \text{ THEN } c\,(CF=\mu), \lambda_1, \lambda_2, \ldots, \lambda_n, w_1, w_2, \ldots, w_n$$

B. Definition of Fuzzy Petri Net

FPN is a promising modeling methodology for expert system [2], [6], [12]. A GFPN structure is defined as a 8-tuple [2]

$$\text{FPN} = (P, T, D, I, O, f, \alpha, \beta) \tag{1}$$

where

$P=\{p_1, p_2, \ldots, p_n\}$	set of places;
$T=\{t_1, t_2, \ldots, t_m\}$	set of transitions;
$D=\{d_1, d_2, \ldots, d_n\}$	set of propositions;
$I(O): T \to P^\infty$	input (output) function which defines a mapping from transitions to bags of places;

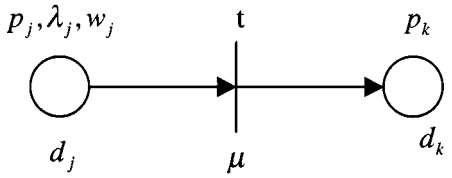


Fig. 1. FPN of Type 1 WFPR in [7].

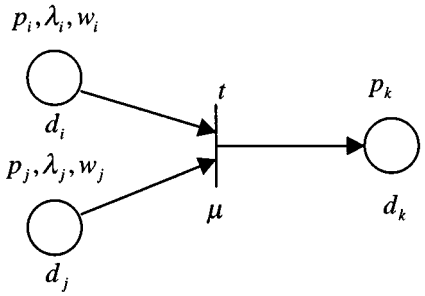


Fig. 2. FPN of Type 2 WFPR in [7].

$f: T \to [0,1]$	association function which assigns a certainty value to each transition;
$\alpha: P \to [0,1]$	association function which assigns a real value between zero to one to each place;
$\beta: P \to D$	bijective mapping between the proposition and place label for each node.

$P \cap T \cap D = \phi, |P| = |D|$.

In order to capture more information of the WFPRs, the FPN model has been enhanced to include a set of threshold values and weights, it consists of a 13-tuple [7]

$$\text{FPN} = (P, T, D, Th, I, O, F, W, f, \alpha, \beta, \gamma, \theta) \tag{2}$$

where

$Th=\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$	set of threshold values;
$F=\{f_1, f_2, \ldots f_s\}$	set of fuzzy sets;
$W=\{w_1, w_2, \ldots, w_r\}$	set of weights of WFPRs;
$\alpha: P \to F$	association function which assigns a fuzzy set to each place;
$\gamma: P \to \text{Th}$	association function which defines a mapping from places to threshold values.

The definitions of P, T, D, I, O, f and β are the same as above. Each proposition in the antecedent is assigned a threshold value, and $\theta: P \to W$ is an association function which assigns a weight to each place.

C. Mapping WFPRs into FPN

The mapping of the three types of weighted fuzzy production rules into the FPNs in [7] are shown in Figs. 1, 2, and 3, respectively. For example, a rule of Type 2 may be represented as

R : IF d_1 AND d_2 AND $\cdots$ AND d_n THEN $c\,(CF=\mu), \gamma(p_j)=\lambda_j, \theta(p_j)=w_j, \alpha(p_j)=f_j, j=1,2,\ldots,n$ (tokens representing fuzzy sets of given facts).

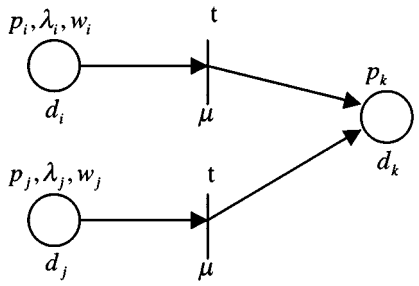


Fig. 3. FPN of Type 3 WFPR in [7].

III. Adaptive Fuzzy Petri Net

FPN [7] can represent WFPRs perfectly. But it can not adjust itself according to the knowledge updating. In another word, it has not learning ability. In this paper, we introduce the conception "adaptive" into FPN, the proposed model is called AFPN.

A. Definition of AFPN

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 [2]. $Th : T \to [0, 1]$ is the function which assigns a threshold value λ_i from zero to one to transition t_i. $Th = \{\lambda_1, \lambda_2, \ldots \lambda_m\}$. $W = W_I \cup W_O$. $W_I : I \to [0, 1]$ and $W_O : O \to [0, 1]$, are sets of input weights and output weights which assign weights to all the arcs of a net.

B. Mapping WFPR into AFPN

The mappings of the three types of WFPR into the AFPNs are shown as Figs. 4, Section III-B, and 5 respectively. The three types of WFPR may be represented as follows.

Type 1: A Simple Fuzzy Production Rule

$$R: \text{IF } a \text{ THEN } c \; Th(t) = \lambda, W_O(t, p_j) = \mu, W_I(p_i, t) = w$$

Type 2: A Composite Conjunctive Rule

$$R: \text{IF } a_1 \text{ AND } a_2 \text{ AND } \cdots \text{ AND } a_n \text{ THEN } c,$$
$$Th(t) = \lambda,$$
$$W_O(t, p_j) = \mu, W_I(p_i, t) = w_i, i = 1, \ldots, n$$

Type 3: A Composite Disjunctive Rule

$$R: \text{IF } a_1 \text{ OR } a_2 \text{ OR } \cdots \text{ OR } a_n \text{ THEN } c,$$
$$Th(t_i) = \lambda_i, W_O(t_i, p_j) = \mu, W_I(p_j, t_i) = w_i, i = 1, \ldots, n$$

The mapping between AFPN and WFPR may be understood as each transition corresponds to a simple rule, composite conjunctive rule or a disjunctive branch of a composite disjunctive rule; each place corresponds to a proposition (antecedent or consequent).

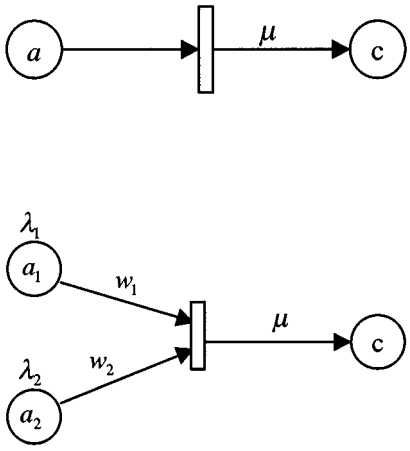


Fig. 4. AFPN of Type 1 WFPR.

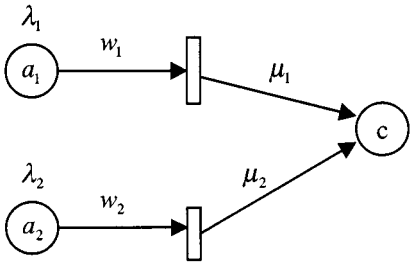


Fig. 5. AFPN of Type 3 WFPR.

C. Fuzzy Reasoning Using AFPN

Firstly, we give some basic definitions which are useful to explain the transition firing rule of AFPN.

Definition 2 (Source Places, Sink Places): A place p is called a source place if it has no input transitions. It is called a sink place if it has no output transitions.

A source place corresponds to a precondition proposition in WFPR, and a sink place corresponds to a consequent. For example, in Fig. 6, P_1, P_2, P_3 are source places, P_6 is a sink place.

Definition 3 (Route): Given a place p, a transition string $t_1 t_2 \ldots t_n$ is called a route to p if p can get a token through firing this transition string in sequence from a group of source places. If a transition string fire in sequence, we call the corresponding route *active*.

For a place p, it is possible that there are more than one route to it. For example, in Fig. 6, $t_1 t_3 t_4$ is a route to P_6, t_2 is another route to it. Let $I(t) = \{p_{i1}, p_{i2}, \ldots, p_{in}\}, w_{I1}, w_{I2}, \ldots, w_{In}$ the corresponding input weights to these places, $\lambda_1, \lambda_2, \ldots, \lambda_n$ thresholds. Let $O(t) = \{p_{o1}, p_{o2}, \ldots, p_{om}\}$, and $w_{o1}, w_{o2}, \ldots, w_{om}$ the corresponding output weights to these places.

We divide the set of places P into three parts $P = P_{\text{UI}} \cup P_{\text{int}} \cup P_O$, where P is the set of places of AFPN; $P_{\text{UI}} = \{p \in P \mid {}^{\cdot}p = \emptyset\}, p \in P_{\text{UI}}$ is called a user input place; $P_{\text{int}} = \{p \in P \mid {}^{\cdot}p \neq \emptyset \text{ and } p^{\cdot} \neq \emptyset\}, p \in P_{\text{int}}$ is called an interior place; $P_O = \{p^{\cdot} = \emptyset\}, p \in P_O$ is called an output place. In this paper, $\emptyset$ is an empty set.

Definition 4: The marking of a place $m(p)$ is defined as the certainty factor of the token in it.

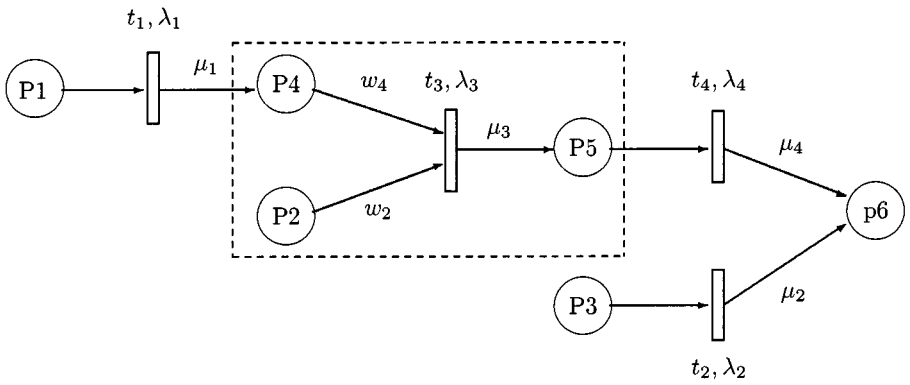


Fig. 6. AFPN of Example 1.

Definition 5: $\forall t \in T, t$ is enabled if $\forall p_{Ij} \in I(t), m(p_{Ij}) > 0, j = 1, 2, \ldots, n$.

Definition 6: When t is enabled, it produces a new certainty factor $CF(t)$

$$CF(t) = \begin{cases} \sum_j m(p_{Ij}) \cdot w_{Ij}, & \sum_j m(p_{Ij}) \cdot w_{Ij} > Th(t) \\ 0, & \sum_j m(p_{Ij}) \cdot w_{Ij} < Th(t). \end{cases}$$

We may use a continuous function $CF(t)(x)$ to approximate $CF(t)$

$$CF(t)(x) = x \cdot F(x)$$

where

$$x = \sum_j m(p_{Ij}) \cdot w_{Ij}$$

where $F(x)$ is a sigmoid function which approximates the threshold of t

$$F(x) = 1 \Big/ \left(1 + e^{-b(x - \mathrm{Th}(t))}\right)$$

where b is an instant. If b is big enough, when $x > Th(t), e^{-b(x-\mathrm{Th}(t))} \approx 0$, then $F(x) \approx 1$, and when $x < Th(t), e^{-b(x-\mathrm{Th}(t))} \to \infty$, then $F(x) \approx 0$.

Definition 7: If $x > Th(t)$, transition t fires, at the same time, token transmission takes place.

1) If a place p_{ok} only has one input transition t, a new token with certainty factor $w_{ok} \cdot CF(t)$ is put into each output place $p_{ok}, k = 1, 2, \ldots, m$, and all tokens in $p_{Ij}, j = 1, 2, \ldots, n$ are removed.
2) If a place p_{ok} has more than one input transitions (as Fig. 5), and more than one of them fire, i.e. more than one routes are active at the same time, then the new certainty factor of p_{ok} is decided by the center of gravity of the fired transitions

$$m(p_{ok}) = \frac{\sum_j [w_{oj} \cdot CF(t_j)]}{\sum_j w_{oj}}$$

where t_j fires, $t_j \in {}^{\cdot}p_{ok}$.

According to above definitions, a transition t is enabled if all its input places have tokens, if the certainty factor produced by it is greater than its threshold, then t fires, so an AFPN can be implemented. Thus, through firing transitions, certainty factors can be reasoned from a set of known antecedent propositions to a set of consequent propositions step by step.

Let $T_{\mathrm{initial}} = \{t \in T \,|\, {}^{\cdot}t \cap P_{\mathrm{UI}} \neq \emptyset \text{ and } {}^{\cdot}t \cap P_{\mathrm{int}} = \emptyset\}, t \in T_{\mathrm{initial}}$ is called an initially enabled transition.

Let $T_{\mathrm{current}} = \{t \in T_{\mathrm{initial}} \,|\, \forall p_i \in {}^{\cdot}t, m(p_i) > 0, \text{ and } CF(t) > Th(t)\}, t \in T_{\mathrm{current}}$ is called a current enabled transition.

Fuzzy Reasoning Algorithm

INPUT: the certainty factors of a set of antecedent propositions (correspond to P_{UI} in AFPN)

OUTPUT: the certainty factors of a set of consequence propositions (correspond to $P_{\mathrm{int}} \cup P_O$ in AFPN)

Step 1) Build the set of user input places P_{UI}.
Step 2) Build the set of initially enabled transitions T_{initial}.
Step 3) Find current enabled transitions T_{current} according to *Definition 5*.
Step 4) Calculate new certainty factors produced by fired transitions according to *Definition 6*.
Step 5) Make token transmission according to *Definition 7*.
Step 6) Let $T = T - T_{\mathrm{current}}, P = P - {}^{\cdot}T_{\mathrm{current}}$.
Step 7) Go to *Step 3* and repeat, until $T_{\mathrm{current}} = \emptyset$.

IV. Knowledge Learning and AFPN Training

In [13], we developed a weights learning algorithm under following conditions.

1) It is necessary to know the certainty factors of all output places (i.e. the right hand of all rules).
2) Only one layer of weights can be learned.
3) For rules of Type 3, if there are more than one transition fire, we must know which input transition is the token contributor to the output place.
4) In case 2 of the Definition 7, error distribution.

These conditions are very strict, because these information in real expert systems may be not available. In this paper we will relax these conditions to more general cases. The main idea is that all layer weights can be updated through the back-propagation algorithm if certainty factors of all sink places are given.

Back propagation algorithm

We assume that

- AFPN model of an expert system has been developed;

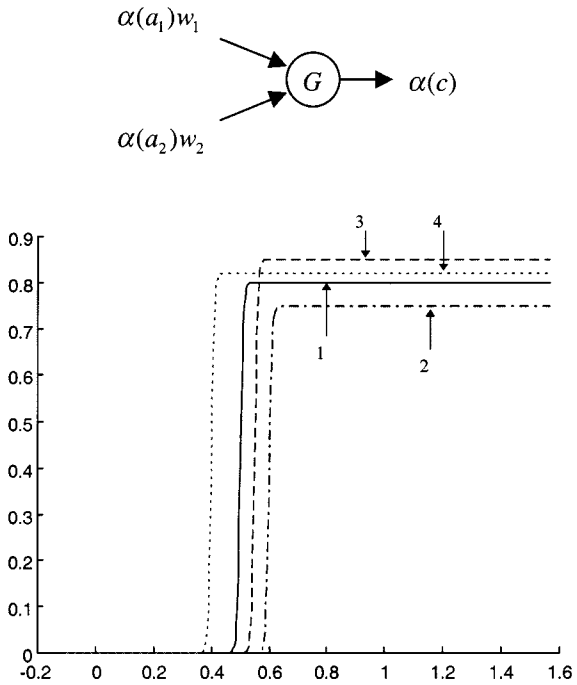


Fig. 7. Sigmiod functions in Example 1.

- in AFPN model, *Th* and W_O are known;
- set of certainty factor values of $P_{\rm UI}$ and P_O is given.

Here we take Type 2 as an illustration to show knowledge learning procedure using AFPN. Type 2 can be translated into an AFPN like Section III-B, this AFPN structure can be translated further into a neural networks-like structure (see Section IV), where G is

$$G(x) := f(x) \cdot x \tag{3}$$

where

$x = W^T\Lambda$	$= \sum_{i=1}^{n} \alpha_i w_i, f(x) = \mu/1 + e^{-b(x-\lambda)}$ sigmoid function;
b	constant which adjust the steepness of $f(x)$;
W	weight vector $W := [w_1, w_2 \dots w_n]^T$;
α	output vector of previous layer, $\Lambda := [\alpha_1, \alpha_2 \dots \alpha_n]^T$.

This continuous function may approximate a logic factor if μ, b and λ are selected suitable values. For example, no. 1 in Fig. 7 has the values as $\mu = 0.8, b = 200$ and $\lambda = 0.5$.

For a place p, there are some learning routes which are from a set of source places to it. The weights in these routes can be trained according the back propagation algorithm developed in this section. Along the selected route Ω, the feedforward propagation process (one hidden layer) is that given any input data U and the fixed weights w_i, the output $O(\Omega)$ can be expressed

$$O(\Omega) = G^{(n)}\left\{W^{(n)}G^{(n-1)}\left[W^{(n-1)}G^{(n-2)}\dots W^{(2)}G^{(1)} \times \left(W^{(1)}U\right)\right]\right\}$$

TABLE I
RESULTS OF AFPN

Group No.	α(P1)	α(P2)	α(P3)	α(P4)	α(P6)	α(P6)
1	0.2190	0.0470	0.6789	0	0	0.3243
2	0.6793	0.9347	0.3835	0.5434	0.5850	0.3055
3	0.5194	0.8310	0.0346	0.4072	0.4517	0.2359
4	0.0535	0.5297	0.6711	0	0	0.3206
5	0.0077	0.3834	0.0668	0	0	0
6	0.4175	0.6868	0.5890	0	0	0.0279
7	0.9304	0.8462	0.5269	0.7443	0.6647	0.3472
8	0.0920	0.6539	0.4160	0	0	0
9	0.7012	0.9103	0.7622	0.5610	0.5867	0.6705
10	0.2625	0.0475	0.7361	0	0	0.3516

where $G^{(k)}\ (k = 1 \dots n)$ is the active function of the k_th layer, $W^{(k)}\ (k = 1 \dots n)$ is the weight of the k_th layer. If the real data is O^*, the output error vector is

$$e_n = O(\Omega) - O^*.$$

Since we do not process the tokens in the output layer, the output layer may be selected as the rule of the center of gravity (see Definition 7), i.e.,

$$G^{(n)}(x) = \frac{x}{\sum_j w_{oj}} \tag{4}$$

The learning algorithm is the same as the backpropagation of multilayer neural networks:

- The weights in output layer is updated as

$$W^{(n)}(k+1) = W^{(n)}(k) - \gamma_i e_n \Lambda^{(n)}$$

where

$\Lambda^{(n)}$	input of the n_th layer;
$\gamma_i > 0$,	adaptive gain;
$W(k)$	weight at the time of k.

$$e_i = e_{i+1}\dot{G}^{(i+1)}(x)W^{(i)} \tag{5}$$

the weights are updated as

$$\begin{aligned} W^{(n-1)}(k+1) &= W^{(n-1)}(k) - \gamma_{n-1}\dot{G}^{(n-1)}e_{n-1}\Lambda^{(n-1)} \\ &\vdots \\ W^{(2)}(k+1) &= W^{(2)}(k) - \gamma_2\dot{G}^{(2)}e_2\Lambda^{(2)} \\ W^{(1)}(k+1) &= W^{(1)}(k) - \gamma_1\dot{G}^{(1)}e_1\Lambda^{(1)} \end{aligned} \tag{6}$$

where $\dot{G}$ is the derivative of the nonlinear function G.

$$\begin{aligned} \dot{G} &:= \frac{d}{dx}\left[\frac{\mu \cdot x}{1 + e^{-b(x-\lambda)}}\right] \\ &= \frac{\mu x b e^{-b(x-\lambda)}}{\left(1 + e^{-b(x-\lambda)}\right)^2} + \frac{\mu}{1 + e^{-b(x-\lambda)}}. \end{aligned} \tag{7}$$

The justification of backpropagation can be found in [11]. Finally, we summarize the learning algorithm of AFPN as

Step 1) Select a set of initial weight values.
Step 2) For each set of input data, find all active routes, and mark them $\Omega_1, \Omega_2, \ldots, \Omega_k$.
Step 3) Following each active route, according to the reasoning algorithm, calculate the corresponding output.
Step 4) Set the difference between the idea output and the calculated output as the error e, select γ_n use (6) to adjust the weights on these routes.

V. Simulation

In this section, two typical examples are selected to show the results in the prior sections.

Example 1: $P1, P2, P3, P4, P5$ and $P6$ are related propositions of an expert system Γ_1. Between them there exist the following weighted fuzzy production rules

$R1$: IF $P1$ THEN $P4$ (λ_1, μ_1)
$R2$: IF $P2$ AND $P4$ THEN $P5$ $(w_2, w_4, \lambda_3, \mu_3)$
$R3$: IF $P3$ OR $P5$ THEN $P6$ $(\lambda_2, \lambda_4, \mu_2, \mu_4)$.

This example includes all the three types of rules, in which $R1$ is a simple WFPR, $R2$ is a composite conjunctive one, and $R3$ is a composite disjunctive one. We want to show the fuzzy reasoning and the weights learning algorithm.

First, based on the translation principle, we map Γ_1 into an AFPN as follows (shown as Fig. 6)

$$\text{AFPN}_1 = \{P, T, D, I, O, \alpha, \beta, Th, W\}$$

where $P = \{p_1, p_2, p_3, p_4, p_5, p_6\}, T = \{t_1, t_2, t_3, t_4\}, D = \{P1, P2, P3, P4, P5, P6\}$, $Th = \{\lambda_1, \lambda_2, \lambda_3, \lambda_4\}, W_I = \{w_2, w_4\}, W_O = \{\mu_1, \mu_2, \mu_3, \mu_4\}$.

We have three input propositions ($P1, P2$ and $P3$) and three consequence propositions ($P4, P5$ and $P6$). The data are given as

$$\begin{aligned} &\mu_1 = 0.80, \quad \mu_2 = 0.75, \quad \mu_3 = 0.85, \quad \mu_4 = 0.82 \\ &\lambda_1 = 0.50, \quad \lambda_2 = 0.60, \quad \lambda_3 = 0.55, \quad \lambda_4 = 0.40 \\ &w_4 = 0.63, \quad w_2 = 0.37 \end{aligned}$$

We use four sigmoid functions as

$$F_i(x) := \frac{\mu_i}{1 + e^{-b_i(x - \lambda_i)}}, \quad i = 1, 2, 3, 4$$

to approximate the four thresholds $\lambda_1, \lambda_2, \lambda_3, \lambda_4$, the steepness b_i are selected as 200 (see Fig. 7). Especially, for the transition t_3, the argument of function $G(x)$ is

$$x = \alpha(P4)w_4 + \alpha(P2)w_2.$$

Using fuzzy reasoning algorithm, a set of output data (certainty factors of consequence propositions) can be calculated according to the input data (certainty factors of antecedent propositions). Table I gives the results of AFPN.

One can see that some data are 0. This means that the corresponding thresholds were not passed. For example, in Group 1,

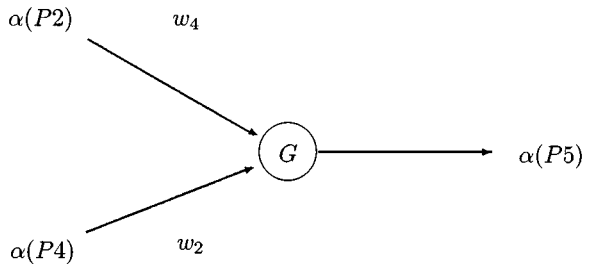


Fig. 8. The neural network translation of the learning part in Example 1.

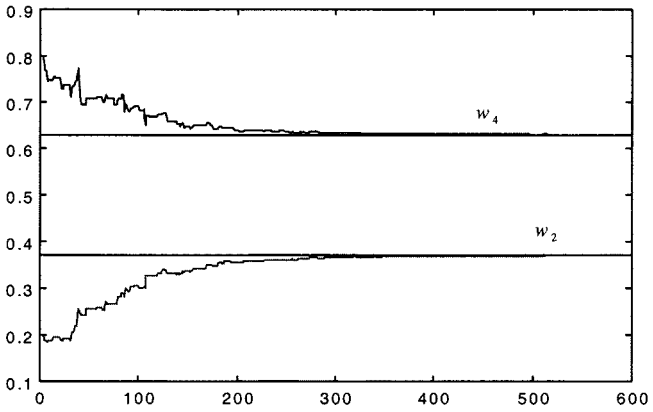


Fig. 9. Single layer learning results of Example 1.

$\alpha(P1) = 0.2190$, the threshold λ_1 is 0.50. Since $\alpha(P1) < \lambda_1$, transition t_1 cannot fire, so the output certainty factor is $\alpha(P4)$ is 0. The use a sigmoid function to approximate a threshold means that exact zero is impossible to get (for example, 0.0001). But if the steepness coefficient is small enough, the sigmoid function can approximate the threshold with good accuracy.

If the weights are unknown, neural networks technique may be used to estimate the weights. The learning part of the AFPN (see the part in the dashed box in Fig. 6) may be formed as a standard single layer neural networks (see Fig. 8). Assume the ideal weights are

$$W_4 = 0.63, \quad W_2 = 0.37.$$

The sigmoid function is

$$F_3(x) := \frac{0.85}{1 + e^{-200(x - 0.55)}}. \tag{8}$$

If the inputs $\alpha(P1)$ and $\alpha(P2)$ are given random data from 1 to 0, we can get the real output $\alpha(P5)$ according to the expert system Γ_1. Given any initial condition for w_4 and w_2, put the same inputs to the neural network. The error between the output of neural network $(\alpha'(P5))$ and that of the expert system $\Gamma_1(\alpha(P5))$ can be used to modified the weights, we may use the following learning law

$$\begin{aligned} W(k+1) &= W(k) + \dot{G}\delta e(k)\Lambda(P(k)) \quad \delta > 0 \\ e(k) &:= \alpha(P5)(k) - \alpha'(P5)(k) \end{aligned} \tag{9}$$

where δ is learning rate, a small δ may assure the learning process is stable. Here, we select $\delta = 0.07$.

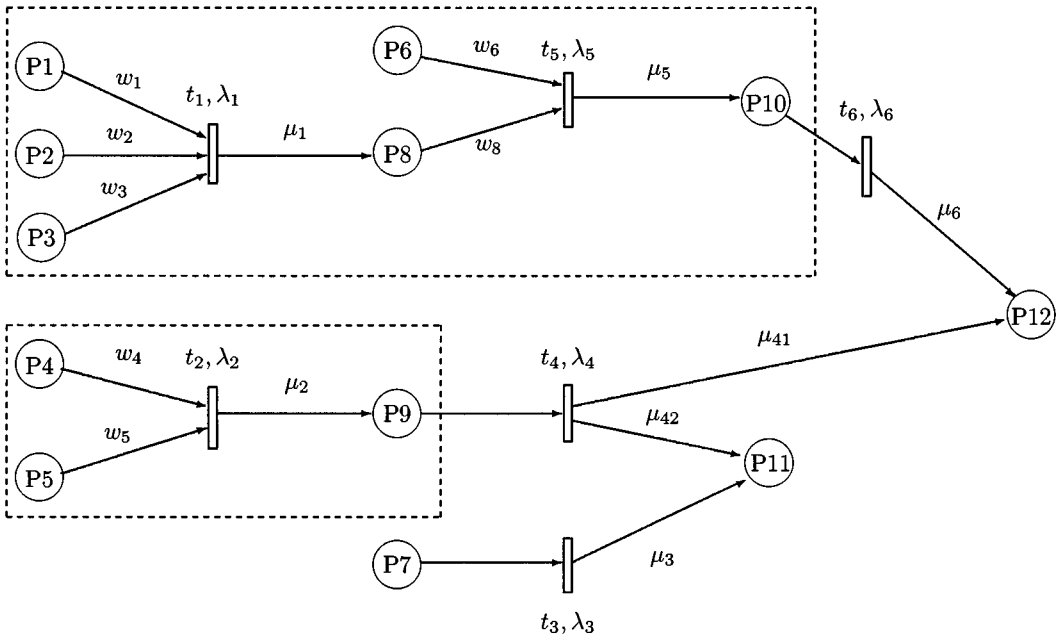


Fig. 10. AFPN of Example 2.

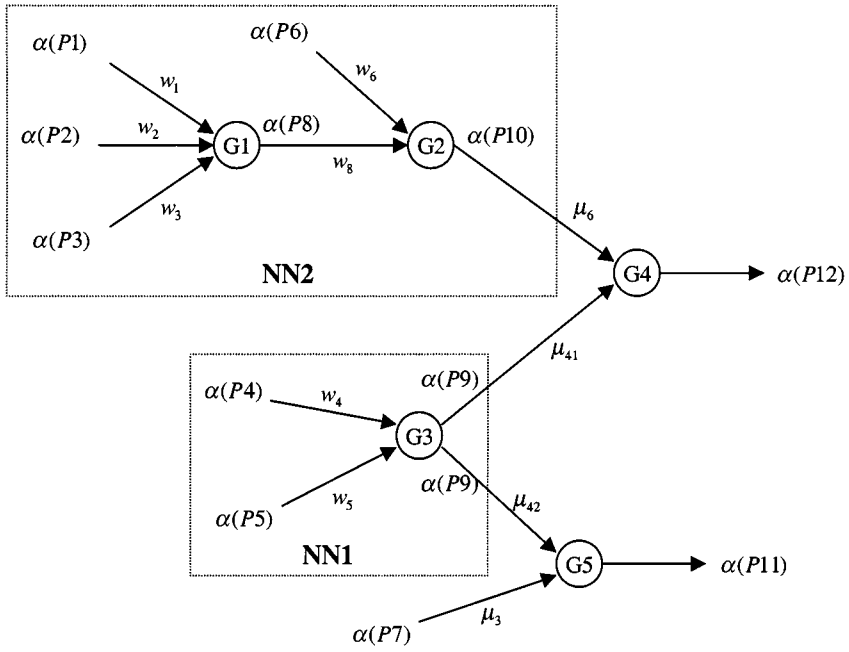


Fig. 11. The neural networks translation of the AFPN in Fig. 12.

$W(k) = [w_2(k), w_4(k)]$, $\Lambda(P(k)) = [\alpha(P2(k)), \alpha(P4(k))]$, and

$$G(x) = \frac{0.85x}{1 + e^{-200(x-0.55)}}.$$

After a training process ($k > 400$), the weights convergence to real values. Fig. 9 shows simulation results.

In this example there is only one learning layer. Example 2 will show a more complicated case where two learning layers (multilayer perceptrons) is used.

Example 2: $P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11$ and $P12$ are related propositions of an expert system Γ_2. There exist the following weighted fuzzy production rules

$R1$: IF $P1$ AND $P2$ AND $P3$ THEN $P8$ $(w_1, w_2, w_3, \lambda_1, \mu_1)$
$R2$: IF $P4$ AND $P5$ THEN $P9$ $(w_4, w_5, \lambda_2, \mu_2)$
$R3$: IF $P6$ AND $P8$ THEN $P10$ $(w_6, w_8, \lambda_5, \mu_5)$
$R4$: IF $P7$ OR $P9$ THEN $P11$ $(\lambda_3, \lambda_4, \mu_3, \mu_{42})$
$R5$: IF $P9$ OR $P10$ THEN $P12$ $(\lambda_4, \lambda_6, \mu_{42}, \mu_6)$

Based on the translation principle, we map Γ_2 into an AFPN (see Fig. 10).

$$\text{AFPN}_2 = \{P, T, D, I, O, \alpha, \beta, Th, W\}$$

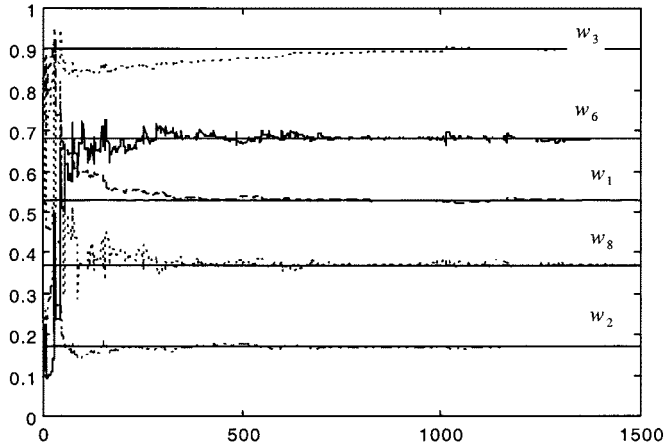


Fig. 12. MLP learning results of Example 2.

where

$$P = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}, p_{12}\}$$
$$T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$$
$$D = \{P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12\}$$
$$Th = \{\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5, \lambda_6\}$$
$$W_I = \{w_1, w_2, w_3, w_4, w_5, w_6, w_8\}$$
$$W_O = \{\mu_1, \mu_2, \mu_3, \mu_{41}, \mu_{42}, \mu_5, \mu_6\}$$

So AFPN model for this expert system may be repressed as in Fig. 10, the two dashed-boxes are the learning parts. This AFPN model may be transferred into a normal neural networks as Fig. 11.

Since the weights of $\mu_{41}, \mu_{42}, \mu_3$ and μ_6 are known, we may simplify this complex neural networks as two sub neural networks: NN1 and NN2. Here sub-networks NN1 is single layer and sub-networks NN2 is multilayer. The neural networks corresponding to G_5 are fixed.

We can train the two networks independently. The original learning error is e_{12}. Because the output function is select as (4)

$$G^{(4)}(x) = \frac{x}{\mu_{41} + \mu_6}, \quad x = \mu_{41}\alpha(P9) + \mu_6\alpha(P10)$$

- In case 1 of Definition 7, if only t_4 fires, then:

$$e_9 = \mu_{41}e_{12}, \quad e_{10} = 0$$

  if only t_6 fires, then:

$$e_{10} = \mu_6 e_{12}, \quad e_9 = 0$$

- In case 2 of Definition 7, when t_4 and t_6 fire at the same time, according to error backpropagation rule (5)

$$e_9 = e_{12} \times \frac{1}{\mu_{41} + \mu_6} \times \mu_{41} = \frac{\mu_{41}}{\mu_{41} + \mu_6} e_{12}$$
$$e_{10} = e_{12} \times \frac{1}{\mu_{41} + \mu_6} \times \mu_6 = \frac{\mu_6}{\mu_{41} + \mu_6} e_{12}$$

The learning algorithms for single layer neural network NN1 is the same as that in Example 1. The adaptive law for multilayer perceptrons NN2 is as in (6). We assume the ideal weights are

$$W_1 = 0.53, \quad W_2 = 0.17, \quad W_3 = 0.9, \quad W_8 = 0.37,$$
$$W_6 = 0.68$$

a set of data about the learning part of the AFPN

$$\lambda_1 = 0.5, \quad \lambda_5 = 0.4, \quad \mu_1 = 0.9, \quad \mu_5 = 0.8,$$
$$b_1 = b_2 = 20$$

Give a set of initial value of the weights

$$w_1(1) = 0.8, \quad w_2(1) = 0.2, \quad w_3(1) = 0.5,$$
$$w_8(1) = 0.7, \quad w_6(1) = 0.1$$

and the learning rate $\delta = 0.5$. The on-line MLP learning results are shown in Fig. 12.

From these two examples, we can see that the fuzzy reasoning algorithm and the back propagation algorithm are very effectively if we do not know the weights of AFPN. After a training process, we can get an excellent input–output mapping of the knowledge system.

VI. Conclusion

This paper introduce a new modified fuzzy Petri net: Adaptive Fuzzy Petri Net (AFPN). It has learning ability as neural networks. So fuzzy knowledge in expert systems can be learned through an AFPN model. The idea proposed in this paper is a new formal way to solve the knowledge learning problem in expert systems. Our ongoing research is to predict expert systems behavior using AFPN framework.

References

[1] H. Scarpelli, F. Gomide, and R. R. Yager, "A reasoning algorithm for high-level fuzzy Petri nets," *IEEE Trans. Fuzzy Syst.*, vol. 4, no. 3, pp. 282–293, 1996.

[2] S. Chen, J. Ke, and J. Chang, "Knowledge representation using fuzzy Petri nets," *IEEE Trans. Knowl. Data Eng.*, vol. 2, no. 3, pp. 311–319, 1990.

[3] C. G. Looney, "Fuzzy Petri nets and applications," in *Fuzzy Reasoning in Information, Decision and Control Systems*, S. G. Tzafestas and A. N. Venetsanopoulos, Eds. Norwell, MA: Kluwer, 1994, pp. 511–527.

[4] A. J. Bugarn and S. Barro, "Fuzzy reasoning supported by Petri nets," *IEEE Trans. Fuzzy Syst.*, vol. 2, no. 2, pp. 135–150, 1994.

[5] K. Hirota and W. Pedrycz, "OR/AND neuron in modeling fuzzy set connectives," *IEEE Trans. Fuzzy Syst.*, vol. 2, no. 2, pp. 151–161, 1994.

[6] W. Pedrycz and F. Gomide, "A generalized fuzzy Petri net model," *IEEE Trans. Fuzzy Syst.*, vol. 2, no. 4, pp. 295–301, 1994.

[7] D. S. Yeung and E. C. C. Tsang, "A multilevel weighted fuzzy reasoning algorithm for expert systems," *IEEE Trans. Syst., Man, Cybern. A*, vol. 28, no. 2, pp. 149–158, 1998.

[8] T. Cao and A. C. Sanderson, "Representation and analysis of uncertainty using fuzzy Petri nets," *J. Intell. Fuzzy Syst.*, vol. 3, pp. 3–19, 1995.

[9] S. M. Chen, "A fuzzy reasoning approach for rule-based systems based on fuzzy logics," *IEEE Trans. Syst., Man, Cybern. B*, vol. 26, no. 5, pp. 769–778, 1996.

[10] M. L. Garg, S. I. Ahson, and P. V. Gupta, "A fuzzy Petri net for knowledge representation and reasoning," *Inf. Process. Lett.*, vol. 39, pp. 165–171, 1991.

[11] M. T. Hagan, H. B. Demuth, and M. Beale, *Neural Network Design*. Boston, MA: PWS, 1996, ch. 11.

[12] D. S. Yeung and E. C. C. Tsang, "Fuzzy knowledge representation and reasoning using Petri nets," *Expert Syst. Applicat.*, vol. 7, pp. 281–290, 1994.

[13] X. Li and F. L. Rosano, "Adaptive fuzzy Petri nets for dynamic knowledge representation and inference," *Expert Syst. Applicat.*, vol. 19, no. 3, 2000.

Xiaoou Li was born in China in 1969. She received the B.S. and the Ph.D. degrees in applied mathematics and electrical engineering from Northeastern University, Shenyang, China, in 1991 and 1995.

From 1995 to 1997, she was a Lecturer in electrical engineering with the Department of Automatic Control, Northeastern University. From 1998 to 1999, she was an Associate Professor of computer science with the Center for Instrumentation Research, National University of Mexico. Since 2000, she has been an Associate Professor of computer science with the Section of Computing, Department of Electrical Engineering, CINVESTAV-IPN, Mexico. Her research interests include Petri net theory and application, neural networks, artificial intelligence, computer integrated manufacturing, and discrete event systems.

Wen Yu (M'99) was born in Shenyang, China, in 1966. He received the B.S. degree from Tsinghua University, Beijing, China, in 1990 and the M.S. and Ph.D. degrees, both in electrical engineering, from Northeastern University, Shenyang, China, in 1992 and 1995, respectively.

From 1995 to 1996, he was a Lecturer with the Department of Automatic Control, Northeastern University. In 1996, he joined CINVESTAV-IPN, Mexico, where he is a Professor with the Departament of Automatic Control. His research interests include adaptive control, neural networks, and industrial automation.

Felipe Lara-Rosano (A'93) received the B.S. degree in civil engineering from the University of Puebla, Mexico, in 1962, the M.S. degree in mechanical and electrical engineering from the National University of Mexico in 1970 and the Ph.D. degree in engineering in the field of operations research from the National University of Mexico in 1973. Also he performed graduate studies at the University of Aachen, Aachen, Germany in the area of industrial engineering and instrumentation.

He joined the Systems Department, Institute of Engineering, National University of Mexico, in 1970 as Associate Researcher. In 1982, he was promoted to Senior Researcher. Additional posts at this University have included: Head of the Graduate Department for Engineering (1991–1993), Head of the Academic Senate for Mathematics, Physics and Engineering (1993–1997), Head of the Department of Computer Science at the Institute for Applied Mathematics and Systems (1997), and Director of the Centre for Instrumentation Research (1998 to present). His research interests include artificial intelligence, expert systems, theoretical and applied cybernetics, neural nets, fuzzy logic, complex systems analysis and modeling and Petri nets and applications. He has published more than 50 international journal papers, book chapters, and conference proceeding papers in his research areas and another 108 research articles in Mexican media. In addition, he has served as member of program committees of 31 scientific meetings.

Dr. Lara-Rosano was listed in the *2000 Marquis Who's Who in the World* and the *2000 Marquis Who's Who in Science and Engineering*. He was recipient of the Outstanding Scholarly Contribution Award from the Systems Research Foundation in 1995 and the Best Paper Award, for the Anticipatory, Fuzzy Semantic, and Linguistic Systems Symposium of the 3rd International Conference on Computing Anticipatory Systems, Liege, Belgium, in 1999. He is a member of the New York Academy of Sciences, the Mexican Academy of Sciences, the Mexican Academy of Engineering, and the Mexican Academy of Technology as well as an honorary doctorate from the International Institute for Advanced Systems Research and Cybernetics. He was elected to the office of president of the Mexican Society for Instrumentation in 1998 and Executive Secretary of the Mexican Academy of Technology in 2000. He is a fellow and board member of International Institute for Advanced Systems Research and Cybernetics.

Available online at www.sciencedirect.com
ScienceDirect
Neurocomputing 71 (2008) 611–619
NEUROCOMPUTING
www.elsevier.com/locate/neucom

Support vector machine classification for large data sets via minimum enclosing ball clustering

Jair Cervantes[a], Xiaoou Li[a], Wen Yu[b,*], Kang Li[c]

[a]Departamento de Computación, CINVESTAV-IPN, A.P. 14-740, Av.IPN 2508, México, D.F. 07360, México
[b]Departamento de Control Automático, CINVESTAV-IPN, A.P. 14-740, Av.IPN 2508, México, D.F. 07360, México
[c]School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, Ashby Building, Stranmillis Road, Belfast BT9 5AH, UK

Available online 9 October 2007

Abstract

Support vector machine (SVM) is a powerful technique for data classification. Despite of its good theoretic foundations and high classification accuracy, normal SVM is not suitable for classification of large data sets, because the training complexity of SVM is highly dependent on the size of data set. This paper presents a novel SVM classification approach for large data sets by using minimum enclosing ball clustering. After the training data are partitioned by the proposed clustering method, the centers of the clusters are used for the first time SVM classification. Then we use the clusters whose centers are support vectors or those clusters which have different classes to perform the second time SVM classification. In this stage most data are removed. Several experimental results show that the approach proposed in this paper has good classification accuracy compared with classic SVM while the training is significantly faster than several other SVM classifiers.

Keywords: Support vector machine; Classification; Large data sets

1. Introduction

There are a number of standard classification techniques in literature, such as simple rule based and nearest neighbor classifiers, Bayesian classifiers, artificial neural networks, decision tree, support vector machine (SVM), ensemble methods, etc. Among these techniques, SVM is one of the best-known techniques for its optimization solution [10,20,29]. Recently, many new SVM classifiers have been reported. A geometric approach to SVM classification was given by [21]. Fuzzy neural network SVM classifier was studied by [19]. Despite of its good theoretic foundations and generalization performance, SVM is not suitable for classification of large data sets since SVM needs to solve the quadratic programming problem (QP) in order to find a separation hyperplane, which causes an intensive computational complexity.

Many researchers have tried to find possible methods to apply SVM classification for large data sets. Generally, these methods can be divided into two types: (1) modify SVM algorithm so that it could be applied to large data sets, and (2) select representative training data from a large data set so that a normal SVM could handle.

For the first type, a standard projected conjugate gradient (PCG) chunking algorithm can scale somewhere between linear and cubic in the training set size [9,16]. Sequential minimal optimization (SMO) is a fast method to train SVM [24,8]. Training SVM requires the solution of QP optimization problem. SMO breaks this large QP problem into a series of smallest possible QP problems, and it is faster than PCG chunking. Dang et al. [11] introduced a parallel optimization step where block diagonal matrices are used to approximate the original kernel matrix so that SVM classification can be split into hundreds of subproblems. A recursive and computational superior mechanism referred as adaptive recursive partitioning was proposed in [17], where the data are recursively subdivided into smaller subsets. Genetic programming is able to deal with large data sets that do not fit in main memory [12]. Neural networks technique can also be applied for SVM to simplify the training process [15].

*Corresponding author. Tel.: +52 55 5061 3734; fax: +52 55 5747 7089.
E-mail address: yuw@ctrl.cinvestav.mx (W. Yu).

0925-2312/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2007.07.028

For the second type, clustering has been proved to be an effective method to collaborate with SVM on classifying large data sets. For examples, hierarchical clustering [31,1], k-means cluster [5] and parallel clustering [8]. Clustering-based methods can reduce the computations burden of SVM, however, the clustering algorithms themselves are still complicated for large data set. Rocchio bundling is a statistics-based data reduction method [26]. The Bayesian committee machine is also reported to be used to train SVM on large data sets, where the large data set is divided into m subsets of the same size, and m models are derived from the individual sets [27]. But, it has higher error rate than normal SVM and the sparse property does not hold.

In this paper, a new approach for reducing the training data set is proposed by using minimum enclosing ball (MEB) clustering. MEB computes the smallest ball which contains all the points in a given set. It uses the core-sets idea [18,3] to partition input data set into several balls, named k-balls clustering. For normal clustering, the number of clusters may be predefined, since determining the optimal number of clusters may involve more computational cost than clustering itself. The method of this paper does not need the optimal number of clusters, we only need to partition the training data set and to extract support vectors with SMO. Then we remove the balls which are not support vectors. For the remaining balls, we apply de-clustering technique, and classify it with SMO again, then we obtain the final support vectors. The experimental results show that the accuracy obtained by our approach is very close to the classic SVM methods, while the training time is significantly shorter. The proposed approach can therefore classify huge data sets with high accuracy.

2. MEB clustering algorithm

MEB clustering proposed in this paper uses the concept of core-sets. It is defined as follows.

Definition 1. The ball with center c and radius r is denoted as $B(c,r)$.

Definition 2. Given a set of points $S=\{x_1,\ldots,x_m\}$ with $x_i\in\mathbb{R}^d$, the *MEB* of S is the smallest ball that contains all balls and also all points in S, it is denoted as $MEB(S)$.

Because it is very difficult to find the optimal ball $MEB(S)$, we use an approximation method which is defined as follows.

Definition 3. $(1+\varepsilon)$-approximation of $MEB(S)$ is denoted as a ball $B(c,(1+\varepsilon)r)$, $\varepsilon>0$ with $r\geqslant r_{MEB(S)}$ and $S\subset B(c,(1+\varepsilon)r)$.

Definition 4. A set of points Q is a core-set of S if $MEB(Q)=B(c,r)$ and $S\subset B(c,(1+\varepsilon)r)$.

For clustering problem, there should be many balls in the data set S. So the definition of $(1+\varepsilon)-$approximation of $MEB(S)$ is modified as:

Definition 5. In clustering, $(1+\varepsilon)-$approximation of $MEB(S)$ is denoted as a set of k balls B_i $(i=1\cdots k)$ containing S, i.e., $S\subset B_1\cup B_2\cup\cdots\cup B_k$.

In other words, given $\varepsilon>0$, a subset Q, is said to be a $(1+\varepsilon)-$approximation of S for clustering if $MEB(Q)=\cup_{i=1}^{k}B_i(c_i,(1+\varepsilon)r_i)$ and $S\subset\cup_{i=1}^{k}B_i(c_i,(1+\varepsilon)r_i)$, i.e., Q is a $(1+\varepsilon)-$approximation with an expansion factor $(1+\varepsilon)$.

Now we consider a finite set of elements $X=\{x_1,x_2,\ldots,x_n\}$ with p- dimensional Euclidian space $x_i=(x_{i1},\ldots,x_{ip})^{\mathrm{T}}\in\mathbb{R}^p$. First we randomly select the ball centers in the data set such that they can cover all range of the data. The radius of the ball r is the most important parameter in MEB clustering. How to choose the user-defined parameter is a trade-off. If r is too small, there will be many groups at the end, their centers will be applied for the first stage SVM. The data reduction is not good. Conversely, if r is too large, many objects that are not very similar may end up in the same cluster, some information will be lost. In this paper, we use the following equation:

$$r_{k,j}=(k-1+\mathrm{rand})\frac{x_{\max,j}-x_{\min,j}}{l},\quad k=1\cdots l,\ j=1\cdots p, \tag{1}$$

where l is the number of the balls, rand is a random number in $(0,1)$, $x_{\max,j}=\max_i(x_{ij})$, $i=1\cdots n$, n is the number of the data, $x_{\min,j}=\min_i(x_{ij})$. In order to simplify the algorithm, we use the same r for all balls

$$r=\frac{\sqrt{\sum_{j=1}^{p}(x_{j,\max}-x_{j,\min})^2}}{2l}. \tag{2}$$

If one data is included in more than one ball, it can be in any ball. This does not affect the algorithm, because we only care about the center of the balls.

In most cases, there is no obvious way to select the optimal number of clusters, l. An estimate of l can be obtained from the data using the method of cross-validation, for example, *v-fold cross-validation* algorithm [4] can automatically determine the number of clusters in the data. For our algorithm, we can first guess the support vector number as sv, then $l\approx\frac{2}{3}sv$.

We use Fig. 1 to explain the MEB clustering, $l=3$. We check if the three balls have already included all data, and if not, we enlarge the radius into $(1+\varepsilon)r$. From Fig. 1 we see that A1, B1 and C1 are included in the new balls (dash lines). But A2, B2 and C2 are still outside of the ball, now we enlarge ε until every data in X is inside the balls, i.e.

$$X\subset\bigcup_{i=1}^{l}B(c_i,(1+\varepsilon)r_i).$$

The MEB clustering algorithm of sectioning l balls is as follows:

Step 1: *Use the random sampling method* (1) *to generate l ball centers* $C=\{c_1,\ldots,c_l\}$ and select the ball radius r as in (2).

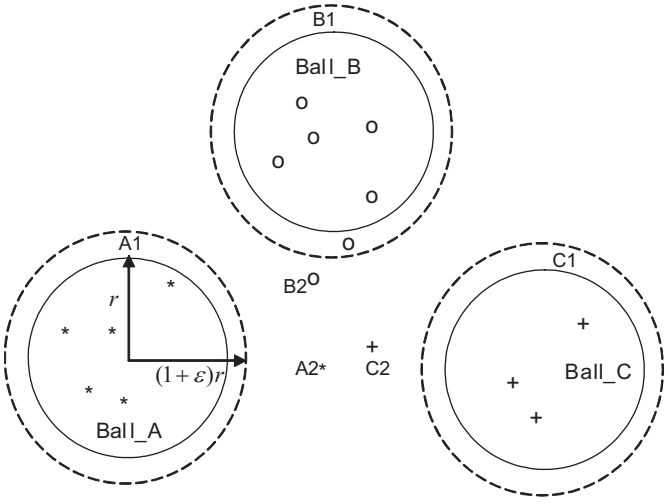


Fig. 1. MED clustering.

Step 2: *For each point* x_i, *we calculate the distance*

$$\varphi_k(x_i) = \|x_i - c_k\|^2$$

calculate the maximum distance $\overline{\varphi}(x_i) = \max_k[\varphi_k(x_i)]$, $k = 1 \ldots l$, $i = 1 \ldots n$. *such that* $\varphi(q)$ *be inside of radius* $B(c_k, (1+\varepsilon)r_k)$ *of each center*, *where*

Step 3: *Complete the clustering, the clusters are* $B_k(c_k, (1+\varepsilon)r)$

Step 4: *If there exists* $\overline{\varphi}(x_i) > (1+\varepsilon)r$, $i = 1 \ldots n$ then *increasing* ε *as*

$$\varepsilon = \varepsilon + \frac{r}{\Delta},$$

where Δ is increasing step, we can use $\Delta = 10$, *and goto step* 2. *Otherwise all data points are included in the balls*, *goto step* 3.

3. SVM classification via MEB clustering

Let (X, Y) be the training patterns set,

$$X = \{x_1, \ldots, x_n\}, \quad Y = \{y_1, \ldots, y_n\}, \quad y_i = \pm 1,$$
$$x_i = (x_{i1}, \ldots, x_{ip})^{\mathrm{T}} \in \mathbb{R}^p. \tag{3}$$

The training task of SVM classification is to find the optimal hyperplane from the input X and the output Y, which maximizes the margin between the classes. By the sparse property of SVM, the data which are not support vectors will not contribute to the optimal hyperplane. The input data sets which are far away from the decision hyperplane should be eliminated, meanwhile the data sets which are possibly support vectors should be used.

Our SVM classification can be summarized in four steps which is shown in Fig. 2: (1) data selection via MEB clustering, (2) the first stage SVM classification, (3) de-clustering, and (4) the second stage SVM classification. The following subsections will give a detailed explanation on each step.

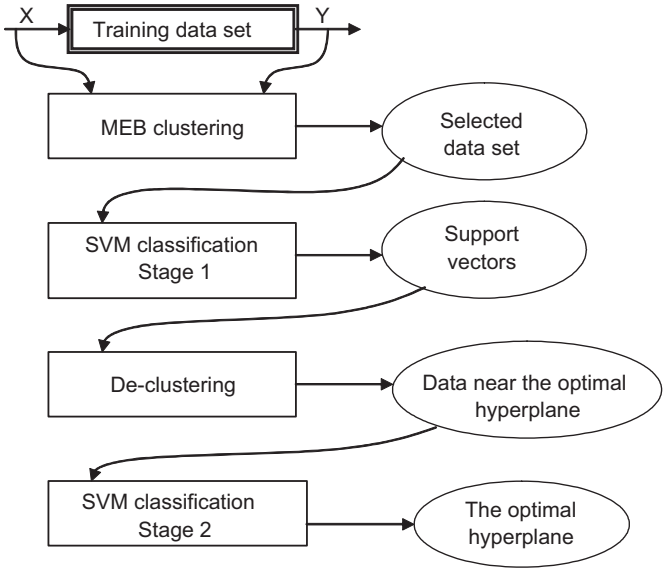


Fig. 2. SVM classification via MEB clustering.

3.1. Data selection via MEB clustering

In order to use SVM, we need to select data from a large data set as the input of SVM firstly. In our approach, we use MEB clustering as data selection method. After MEB clustering, there are l balls with initial radium r and $(1+\varepsilon)-$approximation of MEB(S).

The process of MEB clustering is to find l partitions (or clusters) Ω_i from X, $i = 1, \ldots, l$, $l < n$, $\Omega_i \neq \varnothing$, $\cup_{i=1}^{l} \Omega_i = X$. The obtained clusters can be classified into three types:

(1) clusters with only positive labeled data, denoted by Ω^+, i.e., $\Omega^+ = \{\cup \Omega_i \mid y = +1\}$;

(2) clusters with only negative labeled data, denoted by Ω^-, i.e., $\Omega^- = \{\cup \Omega_i \mid y = -1\}$;

(3) clusters with both positive and negative labeled data (or mix-labeled), denoted by Ω_m, i.e., $\Omega_m = \{\cup \Omega_i \mid y = \pm 1\}$.

Fig. 3(a) illustrates the clusters after MEB, where the clusters with only red points are positive labeled (Ω^+), the clusters with green points are negative labeled (Ω^-) , and clusters A and B are mix-labeled (Ω_m). We select not only the centers of the clusters but also all the data of mix-labeled clusters as training data in the first SVM classification stage. If we denote the set of the centers of the clusters in Ω^+ and Ω^- by C^+ and C^-, respectively, i.e.,

$$C^+ = \{\cup C_i \mid y = +1\} \quad \text{positive labeled centers,}$$

$$C^- = \{\cup C_i \mid y = -1\} \quad \text{negative labeled centers.}$$

Then the selected data which will be used in the first stage SVM classification is the union of C^+, C^- and Ω_m, i.e., $C^+ \cup C^- \cup \Omega_m$. In Fig. 3(b), the red points belongs to C^+, and the green points belong to C^-. It is clear that the data in Fig. 3(b) are all cluster centers except the data in mix-labeled clusters A and B.

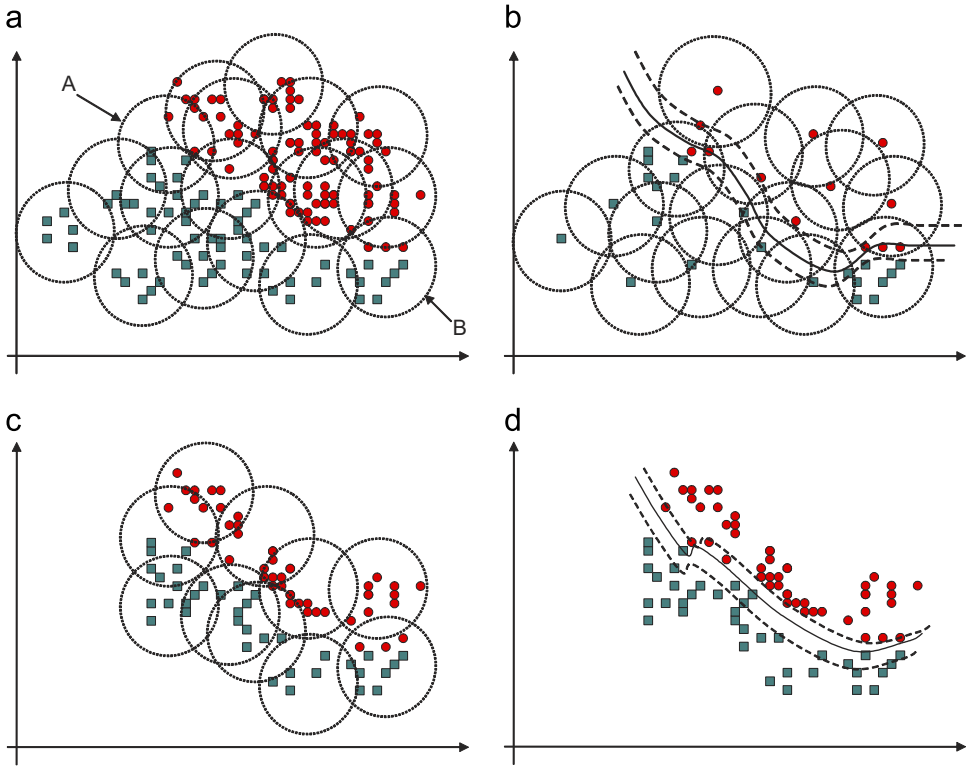


Fig. 3. An example. (a) Data selection; (b) 1st stage SVM; (c) de-clustering; (d) 2nd stage SVM.

3.2. The first stage SVM classification

We consider binary classification. Let (X, Y) be the training patterns set,

$$X = \{x_1, \ldots, x_n\}, \quad Y = \{y_1, \ldots, y_n\},$$

$$y_i = \pm 1, \quad x_i = (x_{i1}, \ldots, x_{ip})^{\mathrm{T}} \in \mathbb{R}^p. \tag{4}$$

The training task of SVM classification is to find the optimal hyperplane from the input X and the output Y, which maximizes the margin between the classes, i.e., training SVM yields to find an optimal hyperplane or to solve the following QP (primal problem),

$$\begin{aligned} &\min_{w,b} \quad J(w) = \frac{1}{2} w^{\mathrm{T}} w + u \sum_{k=1}^{n} \xi_k \\ &\text{subject}: \quad y_k[w^{\mathrm{T}} \varphi(x_k) + b] \geqslant 1 - \xi_k, \end{aligned} \tag{5}$$

where ξ_k is slack variables to tolerate mis-classifications $\xi_k > 0$, $k = 1 \ldots n$, $c > 0$, w_k is the distance from x_k to the hyperplane $[w^{\mathrm{T}} \varphi(x_k) + b] = 0$, $\varphi(x_k)$ is a nonlinear function. The kernel which satisfies the Mercer condition [10] is $K(x_k, x_i) = \varphi(x_k)^{\mathrm{T}} \varphi(x_i)$. Eq. (5) is equivalent to the following QP which is a dual problem with the Lagrangian multipliers $\alpha_k \geqslant 0$,

$$\begin{aligned} &\max_{\alpha} \quad J(\alpha) = -\tfrac{1}{2} \sum_{k,j=1}^{n} y_k y_j K(x_k, x_j) \alpha_k \alpha_j + \sum_{k=1}^{n} \alpha_k \\ &\text{subject}: \quad \sum_{k=1}^{n} \alpha_k y_k = 0, \quad 0 \leqslant \alpha_k \leqslant c. \end{aligned} \tag{6}$$

Many solutions of (6) are zero, i.e., $\alpha_k = 0$, so the solution vector is sparse, the sum is taken only over the non-zero α_k. The x_i which corresponds to non-zero α_i is called a support vector (SV). Let V be the index set of SV, then the optimal hyperplane is

$$\sum_{k \in V} \alpha_k y_k K(x_k, x_j) + b = 0. \tag{7}$$

The resulting classifier is

$$y(x) = \operatorname{sign}\left[\sum_{k \in V} \alpha_k y_k K(x_k, x) + b\right],$$

where b is determined by Kuhn–Tucker conditions.

SMO breaks the large QP problem into a series of smallest possible QP problems [24]. These small QP problems can be solved analytically, which avoids using a time-consuming numerical QP optimization as an inner loop. The memory required by SMO is linear in the training set size, which allows SMO to handle very large training sets [16]. A requirement in (6) is $\sum_{i=1}^{l} \alpha_i y_i = 0$, it is enforced throughout the iterations and implies that the smallest number of multipliers can be optimized at each step is 2. At each step SMO chooses two elements α_i and α_j to jointly optimize, it finds the optimal values for these two parameters while all others are fixed. The choice of the two points is determined by a heuristic algorithm, the optimization of the two multipliers is performed analytically. Experimentally the performance of SMO is very good, despite needing more iterations to converge.

Each iteration uses few operations such that the algorithm exhibits an overall speedup. Besides convergence time, SMO has other important features, such as, it does not need to store the kernel matrix in memory, and it is fairly easy to implement [24].

In the first stage SVM classification we use SVM classification with SMO algorithm to get the decision hyperplane. Here, the training data set is $C^+ \cup C^- \cup \Omega_m$ which has been obtained in the last subsection. Fig. 3(b) shows the results of the first stage SVM classification.

3.3. De-clustering

We propose to recover the data into the training data set by including the data in the clusters whose centers are support vectors of the first stage SVM, we call this process *de-clustering*. Then, more original data near the hyperplane can be found through the de-clustering. The de-clustering results of the support vectors are shown in Fig. 3 (c). The de-clustering process not only overcomes the drawback that only small part of the original data near the support vectors are trained, but also enlarges the training data set size of the second stage SVM which is good for improving the accuracy.

3.4. Classification of the reduced data: the second stage classification

Taking the recovered data as new training data set, we use again SVM classification with SMO algorithm to get the final decision hyperplane

$$\sum_{k\in V_2} y_k \alpha^*_{2,k} K(x_k, x) + b^*_2 = 0, \tag{8}$$

where V_2 is the index set of the support vectors in the second stage. Generally, the hyperplane (7) is close to the hyperplane (8).

In the second stage SVM, we use the following two types of data as training data:

(1) The data of the clusters whose centers are support vectors, i.e., $\cup_{C_i\in V}\{\Omega_i\}$, where V is a support vectors set of the first stage SVM;

(2) The data of mix-labeled clusters, i.e, Ω_m.

Therefore, the training data set is $\cup_{C_i\in V}\{\Omega_i\} \cup \Omega_m$.

Fig. 3(d) illustrates the second stage SVM classification results. One can observe that the two hyperplanes in Fig. 3 (b) and (c) are different but similar.

4. Performance analysis

4.1. Memory space

In the first step clustering the total input data set

$$X = \{x_1, \ldots, x_n\}, \quad Y = \{y_1, \ldots, y_n\},$$
$$y_i = \pm 1, \quad x_i = (x_{i1}, \ldots, x_{ip})^{\mathrm{T}} \in \mathbb{R}^p$$

is loaded into the memory. The data type is float, so the data size is 4 bytes. If we use normal SVM classification, the memory size for the input data should be $4(n \times p)^2$ because of the kernel matrix while the size for the clustering data is $4(n \times p)$. In the first stage SVM classification, the training data size is $4(l+m)^2 \times p^2$, where l is the number of the clusters, m is the number of the elements in the mixed clusters. In the second stage SVM classification, the training data size is $4(\sum_{i=1}^{l} n_i + m)^2 \times p^2$, where n_i is the number of the elements in the clusters whose centers are support vectors. The total storage space of MEB clustering is

$$4(n \times p) + 4p^2 \left[\left(\sum_{i=1}^{l} n_i + m \right)^2 + (l+m)^2 \right]. \tag{9}$$

When n is large (large data sets), n_i, m and $l \ll n$, the memory space by (9) of our approach is much smaller than $4p^2n^2$ which is needed by a normal SVM classification.

4.2. Algorithm complexity

It is clear that without a decomposition algorithm, it is almost impossible for normal SVM to obtain the optimal hyperplane when the training data size n is huge. It is difficult to analyze the complexity of SVM algorithm precisely. This operation involves multiplication of matrices of size n, which has complexity $O(n^3)$.

The complexity of our algorithm can be calculated as follows. The complexity of the MEB is $O(n)$. The approximate complexity of the two SVM training is $O[(l+m)^3] + O[(\sum_{i=1}^{l} n_i + m)^3]$. The total complexity of MEB is

$$O(n) + O[(l+m)^3] + O\left[\left(\sum_{i=1}^{l} n_i + m \right)^3 \right], \tag{10}$$

where l is the total number of cluster, n_i is the number of the elements in the ith clusters whose centers are support vectors, m is the number of the elements in the mixed labeled clusters. Obviously (10) is much smaller than the complexity of a normal SVM $O(n^3)$.

Above complexity grows linearly with respect to the training data size n. The choice of l is very important in order to obtain fast convergence. When n is large, the cost for each iteration will be high, and a smaller l needs more iterations, hence, and will converge more slowly.

4.3. Training time

The training time of the approach proposed in this paper includes two parts: clustering algorithm and two SVMs. The training time of MEB is

$$T_f = \pi \times l \times n \times c_f,$$

where π is the times of $(1+\varepsilon)$-approximation, l is number of clusters, c_e is the cost of evaluating the Euclidian distance.

The training time of SVM can be calculated simply as follows. We assume that the major computational cost comes from multiplication operators (SMO) without considering the cost of the other operators such as memory access. The growing rate of the probability of support vectors is assumed to be constant.

Let $n_m(t)$ be the number of non-support vectors at time t. The probability of the number of support vectors at time t is $F(t)$ which satisfies

$$F(t)=\frac{l+m-n_m(t)}{l+m}, \quad n_m(t)=(l+m)[1-F(t)],$$

where l is the number of the clusters centers and m is the number of the elements in the mixed labeled clusters. The growing rate of the number of support vectors (or decreasing rate of the number of non-support vectors) is

$$h(t)=-\frac{\mathrm{d}[n_m(t)]}{\mathrm{d}t}\frac{1}{n_m(t)}=\frac{\dot{F}(t)}{(l+m)[1-F(t)]}.$$

Since the growing rate is constant $h(t)=\lambda$, the solution of the following ODE:

$$\dot{F}(t)=-\lambda(l+m)F(t)+\lambda(l+m)$$

with $F(0)=0$ is

$$F(t)=1-\mathrm{e}^{-\lambda(l+m)t}.$$

The support vector number of the first stage SVM at time t is $n_{sv1}(t)$, it satisfies

$$n_{sv1}(t)=(l+m)F(t)=(l+m)(1-\mathrm{e}^{-\lambda(l+m)t}), \quad \lambda>0 \qquad (11)$$

and is monotonically increasing. The model (11) can be regarded as a growing model by the reliability theory [14].

The support vector number of the second stage SVM at time t is $n_{\mathrm{sv2}}(t)$, it satisfies

$$n_{\mathrm{sv2}}(t)=(l_1+m)(1-\mathrm{e}^{-\lambda(l_1+m)t}), \quad \lambda>0,$$

where

$$l_1=\sum_{i=1}^{l} n_i+m.$$

We define the final support vector number in each cluster at the first stage SVM as h_i, $i=1\cdots l$. From (11) we know $h_i=(l+m)(1-\mathrm{e}^{-\lambda(l+m)t})$, so

$$t_i=\frac{1}{\lambda(l+m)}\ln\left(\frac{l+m}{l+m-h_i}\right), \quad i=1\ldots l.$$

We define c_1 as the cost of each multiplication operation for SMO. For each interactive step, the main cost is $4(l+m)c_1$. The cost of the optimization at the first stage is

$$\begin{aligned} T_{\mathrm{op}}^{(1)} &= \sum_{i=1}^{l} 4(l+m)c_1\frac{1}{\lambda(l+m)}\ln\left(\frac{l+m}{l+m-h_i}\right) \\ &= \sum_{i=1}^{l}\frac{4}{\lambda}c_1\ln\left(1+\frac{h_i}{l+m-h_i}\right) \\ &\leqslant \sum_{i=1}^{l}\frac{4c_1}{\lambda}\frac{h_i}{l+m-h_i}. \end{aligned}$$

In the second stage,

$$\begin{aligned} T_{\mathrm{op}}^{(2)} &= \sum_{i=1}^{l} 4(l_1+m)c_1\frac{1}{\lambda(l_1+m)}\ln\left(\frac{l_1+m}{l_1+m-h_i}\right) \\ &\leqslant \sum_{i=1}^{l}\frac{4c_1}{\lambda}\frac{h_i}{l_1+m-h_i}. \end{aligned}$$

Another cost of computing is the calculation of kernels. We define c_2 be the cost of evaluating each element of K. In the first stage is

$$T_{\mathrm{ker}}^{(1)}=(l+m)c_2, \quad T_{\mathrm{ker}}^{(2)}=(l_1+m)c_2.$$

The total time for the three approaches are

$$T_2\leqslant \pi\times l\times n\times c_f+\frac{4}{\lambda}c_1\sum_{i=1}^{l}\left[\frac{h_i}{l+m-h_i}+\frac{h_i}{l_1+m-h_i}\right].$$

5. Experimental results

In this section we use four examples to compare our algorithms with some other SVM classification methods. In order to clarify the basic idea of our approach, let us first consider a very simple case of classification and clustering.

Example 1. We generate a set of data randomly in the range of $(0,40)$. The data set has two dimensions $X_i=[x_{i,1},x_{i,2}]$. The output (label) is decided as follows:

$$y_i=\begin{cases}+1 & \text{if } WX_i+b>th,\\ -1 & \text{otherwise}\end{cases} \qquad (12)$$

where $W=[1.2,2.3]^{\mathrm{T}}$, $b=10$, $th=95$. In this way, the data set is linearly separable.

Example 2. In this example, we use the benchmark data which was proposed in IJCNN 2001. It is a neural network competition data set, and available in [25,7]. The data set has 49,990 training data points and 91,701 testing data points, each record has 22 attributes.

Example 3. Recently SVM has been developed to detect RNA sequences [28]. However, long training time is needed. The training data is at "http://www.ghastlyfop.com/blog/tag_index_svm.html/". To train the SVM classifier, a training set contains every possible sequence pairing. This resulted in 475,865 rRNA and 114,481 tRNA sequence pairs. The input data were computed for every

sequence pair in the resulting training set of 486,201 data points. Each record has eight attributes with continuous values between 0 and 1.

Example 4. Another data set of RNA sequence is the training data is available in [22]. The data set contains 2000 data points, each record has 84 attributes with continuous values between 0 and 1.

We will compare our two stages SVM via MEB clustering (named "SMO + MEB"), with LIBSVM [7] (named "SMO") and simple SVM [10] (named "Simple SVM").

For Example 1, we generate 500,000 data randomly whose range and radius are the same as in [31]. The RBF kernel is chosen the same as FCM clustering.

Fig. 4(a) shows "running time" vs "training data size", Fig. 4 (b) shows "testing accuracy" vs "training data size". We can see that for small data set, LIBSVM has less training time and higher accuracy. Our algorithm does not have any advantage. But for large data set, the training time is dramatically shortened in comparison with other SVM implementations. Although the classification accuracy cannot be improved significantly, the testing accuracy is still acceptable.

For Example 2, we use sets 1000, 5000, 12,500, 25,000, 37,500 and 49,990 training data sets. For the RBF kernel

$$f(x,z)=\exp\left(-\frac{(x-z)^{\mathrm{T}}(x-z)}{2r_{ck}^2}\right) \tag{13}$$

we choose $r_{c_k}=r/5$, r is the average of the radius of the clusters, $r=1/l\sum_i^l r_i$. The comparison results are shown in Fig. 5, where the running time vs training data size is (a), and testing accuracies vs training data size is (b). We see that simple SVM has better classification accuracy than our approach. However, the training time is quite long since it works on the entire data set (close to 20,000 s) comparing with our results (less than 400 s).

For Example 3, the comparison results are shown in Table 1.

For Example 4, the comparison results are shown in Table 2.

We can see that for the two ENA sequences, the accuracies are almost the same, but our training time is significantly shorter than that of LIBSVM.

6. Conclusion and discussion

In this paper, we proposed a new classification method for large data sets which takes the advantages of the minimum enclosing ball and the support vector machine (SVM). Our two stages SVM classification has the

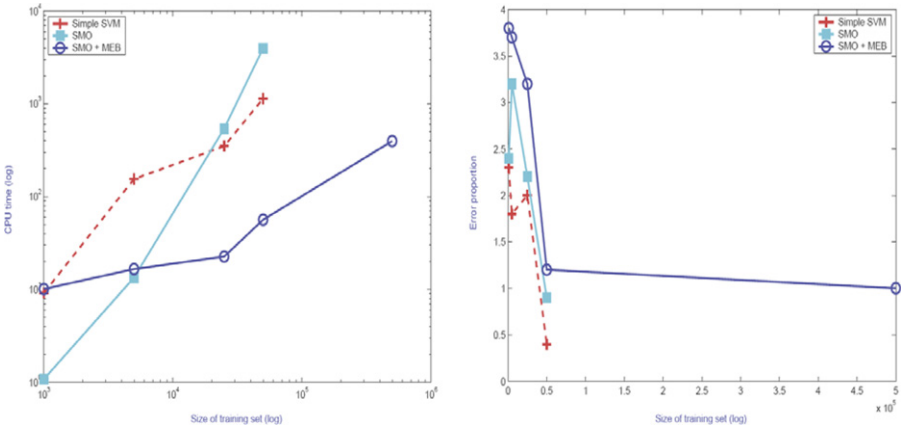


Fig. 4. The running time vs training data size (a) and testing accuracies vs training data size (b).

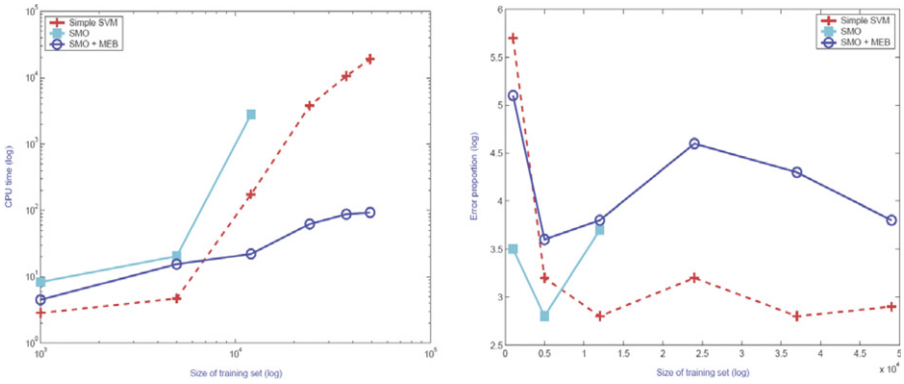


Fig. 5. Comparison with SMO and simple SVM.

Table 1
Comparison with the other SVM classification

RNA sequence 1												
MEB two stages				LIBSVM			SMO			Simple SVM		
#	t	Acc	K	#	t	Acc	#	t	Acc	#	t	Acc
500	45.04	79.6	300	500	0.23	84.88	500	26.125	85.6	500	2.563	85.38
1000	103.6	82.5	300	1000	0.69	85.71	1000	267.19	87.5	1000	9.40	87.21
5000	163.2	85.7	300	5000	10.28	86.40				5000	539.88	88.65
23,605	236.9	88.5	300	23,605	276.9	87.57						

Table 2
Comparison with the other SVM classification

RNA sequence 2												
MEB two stages				LIBSVM			SMO			Simple SVM		
#	t	Acc	K	#	t	Acc	#	t	Acc	#	t	Acc
2000	17.18	75.9	400	2000	8.71	73.15	2000	29.42	78.7	2000	27.35	59.15
2000	7.81	71.7	100									

following advantages compared with the other SVM classifiers:

1. It can be as fast as possible depending on the accuracy requirement.
2. The training data size is smaller than that of some other SVM approaches, although we need twice classifications.
3. The classification accuracy does not decrease because the second stage SVM training data are all effective.

References

[1] M. Awad, L. Khan, F.Bastani, I. L.Yen, An effective support vector machine SVMs performance using hierarchical clustering, in: Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'04), 2004, pp. 663–667.

[3] M. Badoiu, S. Har-Peled, P. Indyk. Approximate clustering via core-sets, in: Proceedings of the 34th Symposium on Theory of Computing, 2002.

[4] P. Burman, A comparative study of ordinary cross-validation, v-Fold cross-validation and the repeated learning-testing methods, Biometrika 76 (3) (1989) 503–514.

[5] J. Cervantes, X. Li, W. Yu, Support vector machine classification based on fuzzy clustering for large data sets, in: MICAI 2006: Advances in Artificial Intelligence, Lecture Notes in Computer Science (LNCS), vol. 4293, Springer, Berlin, 2006, pp. 572–582.

[7] C.-C.Chang, C.-J. Lin, LIBSVM: a library for support vector machines, ⟨http://www.csie.ntu.edu.tw/~cjlin/libsvm⟩, 2001.

[8] P.-H. Chen, R.-E. Fan, C.-J. Lin, A study on SMO-type decomposition methods for support vector machines, IEEE Trans. Neural Networks 17 (4) (2006) 893–908.

[9] R. Collobert, S. Bengio, SVMTorch: Support vector machines for large regression problems, J. Mach. Learn. Res. 1 (2001) 143–160.

[10] N. Cristianini, J. Shawe-Taylor, An Introduction to Support Vector Machines and Other Kernel-based Learning Methods, Cambridge University Press, Cambridge, 2000.

[11] J.-X. Dong, A. Krzyzak, C.Y. Suen, Fast SVM training algorithm with decomposition on very large data sets, IEEE Trans. Pattern Anal. Mach. Intell. 27 (4) (2005) 603–618.

[12] G. Folino, C. Pizzuti, G. Spezzano, GP Ensembles for Large-Scale Data Classification, IEEE Trans. Evol. Comput. 10 (5) (2006) 604–616.

[14] B.V. Gnedenko, Y.K. Belyayev, A.D. Solovyev, Mathematical Methods of Reliability Theory, Academic Press, NewYork, 1969.

[15] G.B. Huang, K.Z. Mao, C.K. Siew, D.-S. Huang, Fast modular network implementation for support vector machines, IEEE Trans. Neural Networks, 2006.

[16] T. Joachims, Making large-scale support vector machine learning practice, in: Advances in Kernel Methods: Support Vector Machine, MIT Press, Cambridge, MA, 1998.

[17] S.-W. Kim, B.J. Oommen, Enhancing prototype reduction schemes with recursion: A method applicable for large data sets, IEEE Trans. Syst. Man. Cybern. B 34 (3) (2004) 1184–1397.

[18] P. Kumar, J.S.B. Mitchell, A. Yildirim, Approximate minimum enclosing balls in high dimensions using core-sets, ACM J. Exp. Algorithmics 8, (January 2003).

[19] C.-T. Lin, C.-M. Yeh, S.-F. Liang, J.-F. Chung, N. Kumar, Support-vector-based fuzzy neural network for pattern classification, IEEE Trans. Fuzzy Syst. 14 (1) (2006) 31–41.

[20] O.L. Mangasarian, D.R. Musicant, Successive overrelaxation for support vector machines, IEEE Trans. Neural Networks 10 (1999) 1032–1037.

[21] M.E. Mavroforakis, S. Theodoridis, A geometric approach to support vector machine (SVM) classification, IEEE Trans. Neural Networks 17 (3) (2006) 671–682.

[22] W.S. Noble, S. Kuehn, R. Thurman, M. Yu, J. Stamatoyannopoulos, Predicting the in vivo signature of human gene regulatory sequences, Bioinformatics 21 (1) (2005) i338–i343.

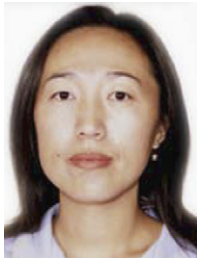
[24] J. Platt, Fast training of support vector machine using sequential minimal optimization, in: Advances in Kernel Methods: Support Vector Machine, MIT Press, Cambridge, MA, 1998.

[25] D. Prokhorov. IJCNN 2001 neural network competition, Ford Research Laboratory, ⟨http://www.geocities.com/ijcnn/nnc_ijcnn01.pdf⟩, 2001.

[26] L. Shih, D.M. Rennie, Y. Chang, D.R. Karger, Text bundling: statistics-based data reduction, in: Proceedings of the 20th International Conference on Machine Learning (ICML-2003), Washington DC, 2003.

[27] V. Tresp, A Bayesian committee machine, Neural Comput. 12 (11) (2000) 2719–2741.
[28] A.V. Uzilov, J.M. Keegan, D.H. Mathews, Detection of non-coding RNAs on the basis of predicted secondary structure formation free energy change, BMC Bioinformatics 173(7) (2006).
[29] Y.S. Xia, J. Wang, A one-layer recurrent neural network for support vector machine learning, IEEE Trans. Syst. Man Cybern. B (2004) 1261–1269.
[31] H. Yu, J. Yang, J. Han, Classifying large data sets using SVMs with hierarchical clusters, in: Proceedings of the 9th ACM SIGKDD 2003, Washington, DC, USA, 2003.

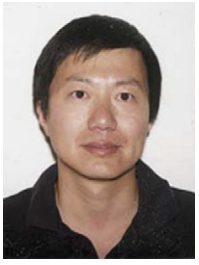
Jair Cervantes received the B.S. degree in Mechanical Engineering from Orizaba Technologic Institute, Veracruz, Mexico, in 2001 and the M.S degree in Automatic Control from CINVESTAV-IPN, México, in 2005. He is currently pursuing the Ph.D. degree in the Department of Computing, CINVESTAV-IPN. His research interests include support vector machine, pattern classification, neural networks, fuzzy logic and clustering.



Xiaoou Li received her B.S. and Ph.D. degrees in applied Mathematics and Electrical Engineering from Northeastern University, China, in 1991 and 1995.

From 1995 to 1997, she was a lecturer of Electrical Engineering at the Department of Automatic Control of Northeastern University, China. From 1998 to 1999, she was an associate professor of Computer Science at the Centro de Instrumentos, Universidad Nacional Autónoma de México (UNAM), México. Since 2000, she has been a professor of the Departamento de Computación, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional (CINVESTAV-IPN), México. During the period from September 2006 to August 2007, she was a visiting professor of School of Electronics, Electrical Engineering and Computer Science, the Queeńs University of Belfast, UK.

Her research interests include Petri net theory and application, neural networks, knowledge based system, and data mining.



Wen Yu He received the B.S. degree from Tsinghua University, Beijing, China in 1990 and the M.S. and Ph.D. degrees, both in Electrical Engineering, from Northeastern University, Shenyang, China, in 1992 and 1995, respectively. From 1995 to 1996, he served as a Lecture in the Department of Automatic Control at Northeastern University, Shenyang, China. In 1996, he joined CINVESTAV-IPN, México, where he is a professor in the Departamento de Control Automático. He has held a research position with the Instituto Mexicano del Petróleo, from December 2002 to November 2003. He was a visiting senior research fellow of Queen's University Belfast from October 2006 to December 2006. He is a also a visiting professor of Northeastern University in China from 2006 to 2008. He is currently an associate editor of Neurocomputing, and International Journal of Modelling, Identification and Control. He is a senior member of IEEE. His research interests include adaptive control, neural networks, and fuzzy Control.

Kang Li is a lecturer in intelligent systems and control, Queen's University Belfast. He received B.Sc. (Xiangtan) in 1989, M.Sc. (HIT) in 1992 and Ph.D. (Shanghai Jiaotong) in 1995. He held various research positions at Shanghai Jiaotong University (1995–1996), Delft University of Technology (1997), and Queen's University Belfast (1998–2002). His research interest covers non-linear system modelling and identification, neural networks, genetic algorithms, process control, and human supervisory control. Dr. Li is a Chartered Engineer and a member of the IEEE and the InstMC.