AN EXPERT SYSTEM FOR DECISION MAKING IN NONMONOTONIC DOMAIN

A. Das , A. Mukherjee & A. Konar
Department of Electronics & Telecommunication Engineering
Jadavpur University, Calcutta 700 032, INDIA.

Abstract : We propose in this paper a new technique of decision making in nonmonotonic domain to tackle databases having uncertain, imprecise, incomplete and inconsistent information. The technique uses a network called Fuzzy Petri Net (FPN) in order to represent a set of propositions and the logical connectivity among them. We have successfully applied the technique to develop an expert system (ES) for criminal investigation.

1. Introduction

A technique for decision making in Petri Nets using fuzzy logic has been proposed by Looney [1] and extended by Konar & Mandal [2,3]. A software realization of the technique to design an ES is presented in this paper. The rule-base of our ES consists of a set of default rules (DR) [4,5] and a set of production rules (PR) [6,7] and the inference engine (IE) has been realized by a FPN. DRs are used to guess about the conclusions to be proved and the FPN is created automatically by detecting the connectivity among the information (database (DB)) using the set of PRs. Fuzzy beliefs are assigned at each proposition of the FPN based on the authenticity level of the sources of information. The beliefs are then updated till an equilibrium condition is reached. Conclusions guessed by the DRs are then searched in the FPN and in case of presence of a number of competitive conclusions, the one having the highest fuzzy belief is declared as the appropriate conclusion. The reasoning for that conclusion is also provided to the user.

The following section provides some fundamental definitions copied from [2] and [8] for reader's convenience. Section 3 describes the schematic architecture of the overall system. The next two sections deal with the data structures and the algorithms used in our ES. Results of complexity analysis of these algorithms are tabulated in section 6.

2. Preliminaries

Def. 1 : Fuzzy Petri Net (FPN) is a geometric structure of four tuples namely i) a set of nodes, ii) a set of arcs, iii) a set of transition bars (TB) and iv) Fuzzy Truth Token (FTT) values at nodes and TBs. Nodes and arcs in a FPN represent information and thier dependence relationship. FTT value at a node denotes the chance of an information being true. FTT values at nodes are useful for token transfer operation, although they have no physical significance. A TB is like a neurone that fires and transmits the FTT value when it exceeds the threshold marked at that TB. A node or an activated TB passes copies of their respective FTT values alor all the arcs emanating from them. Fig 1 shows a typical FPN.

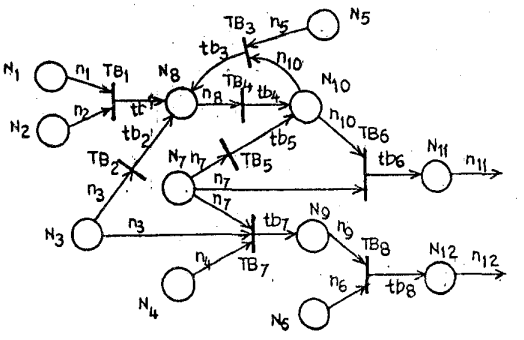

Fig. 1 : A typical FPN

In Fig 1, n_i and tb_j represent FTT values of node N_i and transition bar TB_j respectively.

Def. 2 : A forward path is defined as a connected graph starting from an initial node and terminating to a conclusion node, along which no node is encountered more than once.

Def. 3 : Fuzzy gain of a forward path is defined as the result obtained by ANDing the FTT values carried by all the arcs lying on the forward path.

Def. 4 : Equilibrium or steady-state refers to a state when updated FTT value is equal to the previous FTT value at each node and TB of the FPN.

Fuzzy updation equations : The FTT values in the FPN are updated in parallel using the following equations :
Node updation Equation : If N_p is an independent node (having no input arcs),
then $n_p(t+1) = n_p(t)$
else $n_p(t+1) = \text{Max}\{ tb_k(t) : 1<=k<=u \}$
where u = number of arcs input to the node.
TB updation equation :
Let $x = \text{Min}\{ n_j(t) : 1<=j<=v \}$,
where v = number of arcs input to the TB.
If x>= the threshold marked at the TB, then $tb_q(t+1) = x$ else $tb_q(t+1) = 0$.

Schematic architecture of the ES

Fig 2 shows the schematic architecture of the overall system (copied with modificatios from [8]). The static DB has been organized as a three level tree shown as working memory 1 (WM1) (vide Sec 4). An initial list of suspects is prepared using DRs and a query interface module. These names are used for

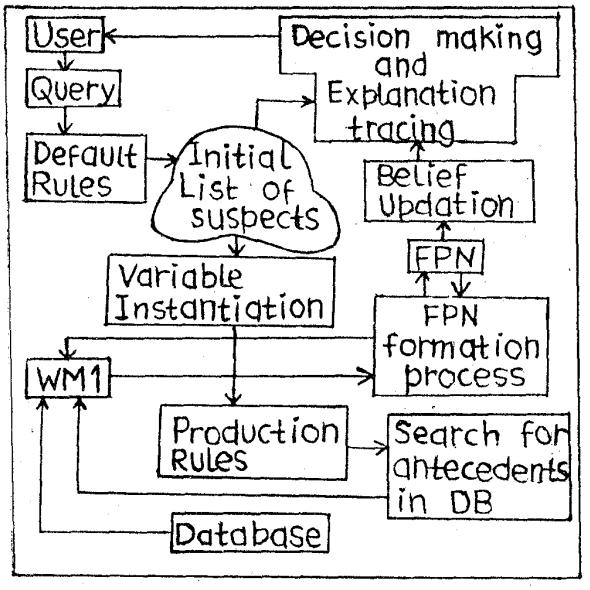

Fig 2 : Schematic architecture of the ES

instantiating variables appearing in the PRs. The FPN is created using the PRs and the static DB. The FTT values of the nodes and TBs of the FPN are then updated in parallel until a steady-state condition is reached. The suspect with the highest FTT value is then declared as the culprit. The most probable reasoning path leading to the conclusion is subsequently provided to the user.

4. Data Structures for the designed ES

Structure of database :
Databases in the ES have been considered as predicates (Pre) having at most two arguments (Arg 1 & Arg 2) and represented by a record structure (vide Fig.3). For a unary predicate, Arg 2 is an underscore.

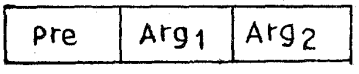

Fig 3 : Structure of Database

Structure of a rule :
Each DR and PR has a format as shown in Fig 4.

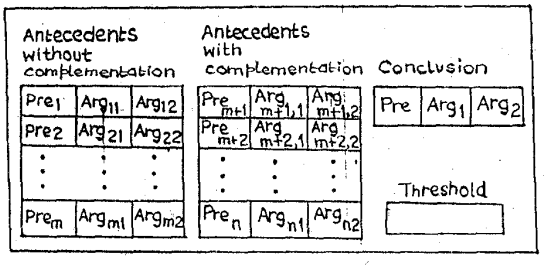

Fig 4 : Structure of DRs and PRs

Example of PR : IF { hasknife(X) AND fingerprint_found_on_knife(X) AND enemy(X,Y) } AND { NOT has_alibi(X) } THEN suspect(X).
This can be represented as in Fig 4.

Structure of a node in FPN :
The software realization of the FPN is as follows. As shown in Fig 5, each node of FPN consists of an identifier(Typ) indicating the type of the node (Node, TB, initial node or conclusion), two pointers L and R to maintain a tree structure, a pointer G to maintain the graph-links, another pointer H to maintain the reverse links, past and next FTT values, threshold (Thr) (significant for TB), Pre, Arg1, Arg2 and number of children(Nc). L and R are used to maintain connectivity of a general tree structure represented in the form of an equivalent binary tree. G and H head two link-lists of pointers each pointing a node in the FPN, as shown in Fig 6.

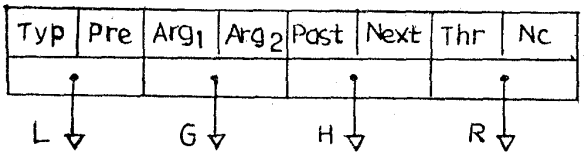

Fig 5 : Structure of a node in FPN

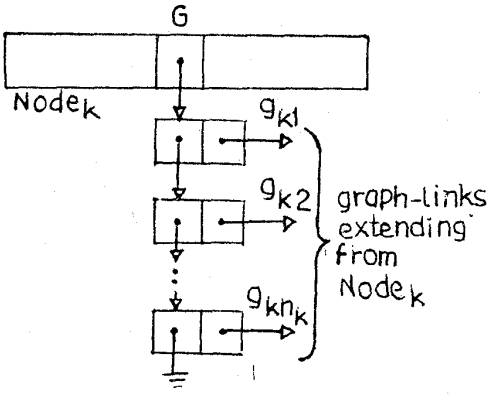

Fig 6 : Graph-links from a node

The datatree :
For efficient searching of DB, the DB of our ES has been organized as a three-level datatree (vide Fig 7). The root holds the starting link of the tree. The second level consists of all the predicates and the third level comprises of various clauses corresponding to each predicate in the second level.

Initially the static DB is represented in the form of the datatree. When PRs fire during FPN formation, the consequence clauses add to the DB thereby increasing the search area for the following PRs. It is,therefore, necessary to dynamically append the datatree with the new clauses during FPN formation.

To illustrate the efficiency of searching let's have P predicates and C clauses against each predicate. So there are P*C clauses in total. To search a particular clause in the datatree requires at most (P+C) comparisons,

whereas sequential search in absence of the daratree calls for (P*C) comparisons in the worst case.

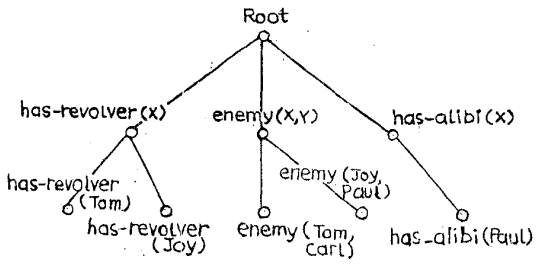

Fig 7 : A typical datatree

5. The design philosophy of the ES

In this section we shall describe the algorithms used in our ES.

Creation of datatree :
1. Create root of datatree.
2. Open datafile (i.e. the file containing the static DB).
3. If eof(datafile) then goto 9.
4. Read clause from datafile.
5. If the predicate is found in the second level of the datatree, then mark the node, else create a new node in the second level to represent the predicate and mark it.
6. Search the clause among the children of the marked predicate.
7. If the clause is not found, then include the clause among the children of the predicate.
8. Goto 3.
9. Close datafile. Stop.

Search for a clause in the datatree :
1. Search the predicate in the second level.
2. If the search fails, then goto 6.
3. Search the clause among the children of the predicate.
4. If the search fails, then goto 6.
5. Search is successful. Stop.
6. Search is not successful. Stop.

Use of default rules :
1. Prepare a list of names of the persons involved in the criminal investigation case.
2 Collect information about the persons and structure those in the form of a datatree.
3. i=1.
4. Pick up DR_i.
5. Instantiate its arguments with names of persons involved.
6. Check whether all antecedents without complementation and no antecedents with complementation are present in the datatree.
7. If so, include the name of the person in the consequence clause of the DR in the list of suspects.
8. If all instantiations are used then goto 9 else with a new set of instantiation goto 6.
9. $i \leftarrow i+1$.
10. If i <= (number of DRs) then goto 4.
11. Stop.

Formation of FPN :
1. i=1.
2. Pick up PR_i.
3. Instantiate variables with names of suspects.
4. Check whether all the antecedents of the PR belong to the datatree.
5. If so , append the FPN by establishing connectivity from antecedent nodes to a TB and from the TB to the consequence node. Also append the datatree by the consequence clause.
6. If all instantiations are used then goto 7 else with a new set of instantiation goto 4.
7. $i \leftarrow i+1$.
8. If i<= (number of PRs) then goto 2.
9. Stop.

Establishing reverse connections :
After a conclusion is reached at the end of the belief revision process, we have to find out the most probable forward path that leads to the conclusion. There are two possibilities. We can find out all forward paths starting from initial nodes and terminating at final nodes and then consider only those paths that lead to the accepted conclusion. But this means much wasted task, since not all forward paths lead to the accepted conclusion. So the second and better strategy is to start from the conclusion node and traverse backward until we reach initial nodes.

With pointers, traversing each step backward requires an exhaustive search on the entire FPN. To avoid such expensive searches, it is expedient to establish reverse connections. If there is a link from node i to node j in the FPN, then we establish a reverse link from node j to node i. Traversing backward in the FPN means traversing forward in the direction of the reverse links.

Finding the most probable reasoning path :
For each conclusion, a conclusion tree is created. The conclusion is at the root of the tree. Each link of the tree represents a reverse connection. The leaves of the tree are, therefore, initial nodes.

Presence of loops poses two problems. First, loops cannot exist in forward paths.Secondly, the conclusion tree grows indefinitely. To avoid, we propose the following strategy.

Whenever a node is created in the conclusion tree, we search for other occurrences of the same node in the tree.If the search succeeds, we check whether the last occurrence is present as a leaf of the subtree headed by each previous occurrence. If at least one such case is detected, the last occurrence is destroyed.

An example for clarification.We Consider the FPN of Fig 1. The conclusion tree for N_{11} is shown in Fig 8. Both N_5 and N_{10} are initially included as children of TB_3 . But a previous occurrence of N_{10} is detected in the tree and the last occurrence is a leaf of the tree headed by the previous occurrence. So the last occurrence is destroyed. In Fig 1, this is equivalent to destroying the loop N_8- TB_4-

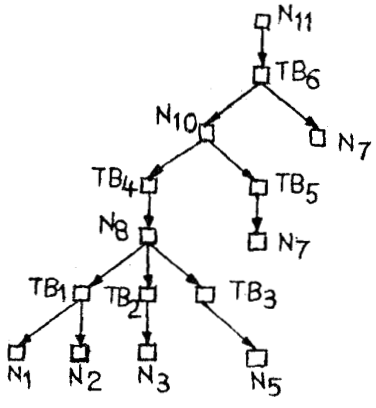

Fig 8 : Conclusion tree for the node N_{11} of Fig 1

N_{10} - TB_3 - N_8 by opening the link from N_{10} to TB_3 .

The conclusion tree gives all forward paths leading to the conclusion. The path with the highest fuzzy gain gives the most probable reasoning path.

6. Complexity analysis results

Let's explain the following notations :

p = number of predicates in datatree
c_i = number of clauses against the ith predicate.
$t = \sum_{i=1}^{p} c_i$ = total number of clauses in the datatree
m = number of persons involved
n = number of suspects
d = number of DRs
r = number of PRs
v = average number of variables per rule
V = maximum number of variables per rule
q = average number of antecedents per rule

During the formation of FPN, the datatree is dynamically appended. p_o and t_o are values of p and t before FPN formation process is started. We make two assumptions :

1. The consequence predicates of the PRs are different from each other.
2. No consequence predicate of the PRs is present in the database before FPN formation.

We can deduce the following expressions for the number of comparisons of the database for different algorithms :

Algorithm	Average Case	Worst Case
Formation of datatree	$\frac{1}{2}[\frac{t^2}{p}+tp-2p]$ if $c_1=c_2=\cdots=c_p=t/p$	$\frac{1}{2}[t^2+t-p^2-1]$
Searching datatree	$\frac{1}{2p}[p^2+t]$	$\frac{1}{p}[p^2+t]$
Use of DRs	$\frac{dqm^v}{16}[4p+m(m+1)]$	$dqm^V[p+m^2]$
Formation of FPN	$\frac{3rqn^v}{32}[4p_o+2(r+1)+n(n+1)]$	$rqn^V[p_o+n^2+\frac{1}{2}(r+1)]$

Table : Number of comparisons in the database for different algorithms

Conclusion

Our ES exploits the combined facilities of structured objects and production rules. This also bridges the gap between relational database and procedural programming. A procedural language like Pascal or C providing pointer facilities and easy mathematical operations is indeed the best choice for our ES.

References :

1. G.C.Looney - Fuzzy Petri Nets for Rule-based Decision Making, IEEE Trans. System, Man and Cybernatics, vol 18, no.1, Jan/Feb 1990.

2. A.Konar & A.K.Mandal, "Stability Analysis of a nonmonotonic Petri Net for diagnostic Systems using Fuzzy Logic", Proc. 33rd. Midwest Symp. on Circuits, Systems and Computers, Aug 1990.

3. A.Konar & A.K.Mandal, "Decision making in nonmonotonic domain using Fuzzy Petri Nets", Proc. Int. Conf. on Automation, Robotics and Comp. Vision, Sep 1990.

4. P.Bernard, "An Introduction to Default Logic", Springer - verlag, 1989, Ch 3,pp 13-30.

5. R.Retier, "A Logic for Default Reasoning", Artificial Intelligence, Vol 13, Apr 1980.

6. B.G.Buchanan & E.H.Shortliffe, Rule-based ES : The MYCIN Expts of the Stanford Univ. heuristic programming projects. Reading MA : Addison Wesley,1984.

7. E.H.Shortliffe, "Computer-based Medical Consultations : "MYCIN" NY : American Elsevier.

8. A.Das & A.Mukherjee : "Nonmonotonic Reasoning in an Expert System for Criminal Investigation", IEEE Student Paper Contest 1991, Organized by IEEE Calcutta Section.