Abstract: This paper briefly describes an open expert system shell that has been under development at the Department of Controls and Instrumentation, Brno University of Technology. It gives account of the most significant functionalities of this system. Two knowledge base representation methods used in this system are described. One of them, based on Bayesian approach, is well in-practice tested and verified. A new method suitable for automatic knowledge base rule weights tuning is proposed and described including the tuning algorithm. This method is now under further development and testing for real world purposes.

Key-Words: Expert systems, Knowledge base tuning

1 Introduction
Expert systems are interactive computer programs for decision support. With the goal of reducing costs, speeding up the decision making process and making it available no matter what the time or place used, they substitute human experts. Expert systems (ES) in general utilize numerous methods for representing expert knowledge – that is acquiring the problem-solving heuristic from the actual human expert, coding it in a proper way and of course using it later for the actual expert system operation. Every knowledge base needs to be thoroughly tuned and verified so that the results given in any computer consultation session are as coherent with the human expert advice as possible [7], [10], [11].

Even though the basic structure of the knowledge base can be built quite easily (it is usually not difficult to determine the basic terms of the problem domain and to link them up based on mutual dependencies), common sense or even expert knowledge may not be enough to set up the weights of such links or may not lead to finding all the necessary connections. The process of creating and tuning a knowledge base can therefore be very long, costly and it is not guaranteed that the results will be satisfactory. Many times this leads into limited use of expert systems.

Our goal is to aid the process of creating the knowledge base by providing tools and methods that make it easy for the expert and the knowledge engineer (expert system programmer) to create and deploy knowledge bases as general and robust as possible.

2 ES shell
Based on the experience from previous development [9], [4] and also practical usage we have introduced an open expert system shell. It operates in two modes.

A) The designer mode gives the knowledge engineer the possibility to code the knowledge base in a graphical (easily readable and editable) way that follows the latest trends in user interface. It lets the engineer decide what method to use for coding the knowledge. Planned is the option for letting the user assign individual methods to be used for individual parts of the respective consultation that are then linked into a sequence for very complex cases (thus we call it hybrid).

B) The expert mode then serves as the final user software product that only “reads” the readily made knowledge base and acts as the interface between the human or machine input and the inference engine (the algorithm that process the input values creating a so called fact base that represents the information about the current problem-solving session). Currently, we only focus on diagnostic expert system behavior.

2.1 Program model
The actual application comprises of “simple” modules sharing a common runtime environment (CRE). Every module operates on a specific field (i.e. one knowledge representation method) and is prepared for both A) and B) modes as described earlier. The common environment also contains the graphics (GUI) engine as well as other standard functionalities. I.e. opening and saving of project (knowledge bases or consultation sessions), the toolbox holding objects for arranging the knowledge bases, object event parsing etc.

Fig. 1 shows the program model and the relationship between the individual modules, the CRE and the application as a whole. Any module that is to be used in
this structure needs to register its exported objects in the CRE toolbox in order for the user (knowledge engineer) be able to use it. Specific events fetched or created during runtime are then passed onto these objects letting the source module handle them independently on the common environment.

Fig. 2 shows the application interface in the designer mode (A) set up for creating a NPS32 type knowledge base (this knowledge representation method is discussed further down in this paper).
The structure shown in Fig. 2 is a simple case of a diagnostic knowledge base. Here, it is used for determining the cause of a TV set malfunction.

3 Knowledge representation methods

In this chapter, the two knowledge representation methods used for building knowledge bases in our expert system shell are described. First is the NPS32 Bayesian based approach [4]. Second is the proposed RESLA method [3] that allowed for the implementation of a gradient knowledge base rule weights optimization algorithm that speeds up the process of tuning and deploying.

3.1 NPS32

Let us consider a basic production rule if $A$ then $C$, where $A$ stands for the rule antecedent, $C$ stands for the consequent. Now consider $A$ holds true. Then

$$ p(C/A) = \frac{p(A/C)p(C)}{p(A)} = \frac{p(C)}{1-p(C)} $$

where $p(C/A)$ is the conditional probability of $C$ assuming that $A$ holds true. $p(A)$ and $p(C)$ are initial probabilities. Similarly $p(C/A)$ applies for the negation of examined consequent $C$. (2), (3), (4) and (5) are probability rates, where $O(C)$ is the a-priori odds of $C$ holding true; $O(C/A)$ is the a-posteriori odds (after the rule has been examined by the expert system’s inference engine); $S$ is the level of sufficiency of a rule and $I$ is the level of rule indispensability

$$ O(C) = \frac{p(C)}{1-p(C)} \quad (2) $$

$$ O(C/A) = \frac{p(C/A)}{1-p(C/A)} \quad (3) $$

$$ S = \frac{p(A/C)}{p(A)} \quad (4) $$

$$ I = \frac{p(A/C)}{p(A)} \quad (5) $$

Using (6) one can enumerate the probability $p(X)$

$$ p(X) = O(X) \frac{1}{O(X)+1} \quad (6) $$

of any term $X$ based on the knowledge of its odds $O(X)$. The expert system user is interested in the a-posterior values (conditional probabilities of rule consequents $C$ when it has been proven that the antecedents either hold true: $A$ or hold false: $\overline{A}$). Depending on these expert system observations (ES inputs) (7) or (8) are then used for enumerating the resulting probabilities of the rule consequents.

$$ O(C/A) = L \cdot O(C) \quad (7) $$

$$ O(C/\overline{A}) = I \cdot O(C) \quad (8) $$

Thus, the knowledge base of the expert system needs to be filled with the rule structure, all terms need to be assigned their initial probabilities and all the rules need to be assigned the values $L$ and $I$. During the session, all antecedents are examined (may hold true $p=1$ or false $p=0$) and resulting probabilities of all consequents can be enumerated.

Note, that this method does not account for any uncertainties in observation $p(A) \in (0;1)$. NPS32 extends the Bayesian concept by introducing probability pairs so that any antecedent or consequent (be that the resulting hypothesis or only an intermediate one) can be rated with a $p \in (0;1)$. The probability $p$ is expressed in terms of the degrees of hypothesis’ truth reliance $x_p$ and falsity reliance $x_n$.

$$ x_p = (1-T); \quad x_n = (1-F) \quad (9) $$

where $T$ and $F$ are the probability pair components, $p,T,F \in <0;1>$ and

$$ p = \frac{F}{F+T}, \quad T = \frac{F(1-p)}{p}, \quad F = \frac{pT}{1-p}. \quad (10) $$

This approach simplifies the computation formulas and allows for simple identification of a contestable statement or absence of relevant information. With the probability values expressed in the form of pairs we can easily model the rules and, or. The probability pair of a consequent is derived from the probability pairs of all the antecedents (11) and (12) where $p_n \equiv (T_n, F_n)$ stands for the consequent and $p_n \equiv (T_n, F_n)$ stands for the $n$th antecedent. Fig. 3 shows the probability pair system in an easily readable way.

$$ (T_1,F_1) \text{ and } (T_2,F_2) \text{ and } ... \text{ and } (T_n,F_n) = (T_c,F_c) = (\max(T_1,\cdots,T_n), \min(F_1,\cdots,F_n)) \quad (11) $$

$$ (T_1,F_1) \text{ or } (T_2,F_2) \text{ or } ... \text{ or } (T_n,F_n) = (T_c,F_c) = (\min(T_1,\cdots,T_n), \max(F_1,\cdots,F_n)) \quad (12) $$
3.2 RESLA method

Here we propose a knowledge base structure that allows for direct tuning of rule weights using a modified back-propagation algorithm. The knowledge base is created in a form of an oriented acyclic graph where the first layer nodes stand for user responses (inputs from an analyzed process), the intermediate layer nodes are used for expressing mutual dependencies and supportive hypothesis (i.e. intermediate consultations results) and the output layer members represent the consultation result-hypothesis.

The RESLA method is a supervised learning algorithm that uses the error function gradient for modifying rule weights. As with neural network back-propagation algorithm it is necessary to have a set of patterns with a set of relevant responses. Different are the “transfer” functions of the rules (13). The condition of rule transfer function differentiability applies. \( y_{ji} \) is the output of the \( j \)-th rule in the \( n \)-th layer, \( w_{ji} \) is its weight, \( x \) is the rule antecedent vector and \( \Theta(x) \) is the AND/OR function.

\[
y_{ji} = w_{ji} \cdot \Theta(x) \quad (13)
\]

The standard AND, OR functions are non-differentiable, therefore we replace them with the t-norm (14) or s-norm (15) respectively. \( y \) is the function output, \( x \), are input variables.

\[
y = \prod_{i} x_{i} \quad (14)
\]

\[
y = \bigcup(x_1, x_2) = x_1 + x_2 - x_1 \cdot x_2 \quad (15)
\]

The rules used within a knowledge base are defined as follows: Logical product (16)

\[
y_{ji}^{n} = w_{ji}^{n} \cdot \prod_{i=1}^{N} \left( \lambda_{ji}^{n-1} \left( y_{i}^{n-1} \right) \right) \quad (16)
\]

where \( w_{ji}^{n} \) is the weight of \( j \)-th rule in the \( n \)-th layer, \( y_{i}^{n-1} \) is the output of the \( i \)-th rule in the \((n-1)\)-th layer, \( N \) is the number of \( j \)-th rule inputs and \( \lambda_{ji}^{n} = 1 - x \) for a negated link; \( \lambda_{ji}^{n} = x \) otherwise. Logical sum (17)

\[
y_{ji}^{n} = w_{ji}^{n} \cdot \bigcup_{i=1}^{N} \text{OR} \left( \lambda_{ji}^{n} \left( y_{i}^{n-1} \right) \right) \quad (17)
\]

where \( \text{OR}(x) \) is a recursive function

\[
\text{OR}(x) = \bigcup(x_1, \bigcup(x_2, \ldots \bigcup(x_{i-1}, x_i))) \quad (18)
\]

The aggregation function (19) provides for the composition of individual rule- sub-tree influences into the probability values of the goal node (goal hypothesis).

\[
y_{j}^{n} = w_{j}^{n} \cdot \bigcup_{i=1}^{N} \left( y_{i}^{n-1} \right) \quad (19)
\]

At the beginning of weight optimization, all the weights \( w \) are set randomly from \( \{0, 1\} \). In every step, one model input is submitted. The knowledge base as a whole executes projection \( \Psi : x \rightarrow y \). \( y = \Psi(x) \), where \( y \) is the output vector, \( x \) is the input vector, \( \Psi \) is the knowledge base system function. The error function is then

\[
e = d - y = d - \Psi(x) \quad (20)
\]

\( d \) is the vector of desired outputs. The immediate quadratic error for one rule output (from (13) and (20)) is then

\[
e_{j} = \frac{1}{2} \sum_{p=1}^{M} \left( d_{j-p} - w_{j-p}^{n} \cdot y_{j} \left( w_{j-p}^{n}, \Theta_{j-p} \left( \lambda_{ji}^{(n-1)} \right) \right) \right)^{2} = \frac{1}{2} \sum_{p=1}^{M} \left( \frac{1}{2} \left( d_{j-p} - w_{j-p}^{n} \cdot \Theta_{j-p} \left( \lambda_{ji}^{(n-1)} \right) \right)^{2} \right) \quad (21)
\]

where \( M \) is the number of examples belonging to the \( j \)-th output, \( p \) is the respective sample. Similarly, the error
for the aggregation rule can be determined. The quality criterion is
\[ e = E[e_j]. \] (22)

The weights are set according to the delta rule
\[ w(k+1) = w(k) - \mu_k \nabla e, \] (23)

\( \mu_k \) is the learning constant (determining the speed of adaptation process), \( k \) is step index. (24) shows the rule (23) rewritten for individual knowledge base components
\[ w_j^n(k+1) = w_j^n(k) - \mu_k \frac{\partial e}{\partial w_j^n}(k). \] (24)

Since the rule function is \( y_j^n = w_j^n \cdot \Theta_j(x) \), we can express the gradient of the error function according to the respective weight \( w \) as
\[ \frac{\partial e^n}{\partial w_j^n} = \frac{\partial e^n}{\partial y_j^n} \cdot \frac{\partial y_j^n}{\partial w_j^n} = \frac{\partial w_j^n}{\partial y_j^n} \cdot \alpha_j^n, \] (25)

here \( \alpha_j^n \) denotes the rule’s inner “potential”. For the output layer, we have
\[ \frac{\partial e_j^n}{\partial w_j^n} = \frac{1}{2} 2(d_j - y_j^n \cdot \Theta_j(x))(-\Theta_j(x)) = -(d_j - y_j^n) \cdot \alpha_j^n. \] (26)

Similarly, the gradients can be expressed for the aggregation rules and for intermediate layers (27)
\[ \frac{\partial e_{j}^{n-1}}{\partial y_j^{n-1}} = \sum_{y_j^{n-1} \rightarrow \Theta_j} \frac{\partial e_j^n}{\partial y_j^n} \cdot w_j^n \cdot \frac{\partial \Theta_j^n(x)}{\partial y_j^{n-1}}. \] (27)

The rule weights for the \( n \)th (output) layer are set according to (28), for intermediate layers according to (29).
\[ w_j^n(t+1) = w_j^n(t) + \mu_t \alpha_j^n(y_j^n - d_j^n). \] (28)
\[ w_j^{n-1}(t+1) = w_j^{n-1}(t) + \mu_t \alpha_j^{n-1} \sum_{y_j^{n-1} \rightarrow \Theta_j} \frac{\partial e_j^n}{\partial y_j^n} \cdot w_j^n \cdot \frac{\partial \Theta_j^n(x)}{\partial y_j^{n-1}}. \] (29)

For individual rule and link types, substitute expressions for \( \frac{\partial \Theta_j^n(x)}{\partial y_j^{n-1}} \) were found. Further mathematical description is beyond the extent of this paper.

3.3 RESLA experimental results

The method was tested on a knowledge base executing three logical functions. The output knowledge base nodes \( Y_{\text{and}}, Y_{\text{or}}, Y_{\text{xor}} \) modeled three log. functions, each of them having three input arguments \( X_1, X_2, X_3 \):

- \( Y_{\text{and}} = X_1 \land X_2 \land X_3 \)
- \( Y_{\text{or}} = X_1 \lor X_2 \lor X_3 \)
- \( Y_{\text{xor}} = \left( \left( X_1 \land X_2 \right) \lor \left( X_1 \land \overline{X_2} \right) \right) \land X_3 \lor \left( \left( X_1 \land \overline{X_2} \right) \lor \left( X_1 \land X_3 \right) \right) \land \overline{X_3} \)

The following figures show the development of the rule weights in the adaptation process.

Fig. 4 \( \mu_k = 0.1 \)

Fig. 5 \( \mu_k = 0.4 \)
Figures 6 and 7 compare the development of the global average error for the knowledge base weights.

\[ \mu_k = 0.1 \]

\[ \mu_k = 0.4 \]

4 Conclusion

In this paper we have described the hybrid open expert system shell that has been developed at the Department of Controls and Instrumentation, Brno University of Technology. It is intended as a tool for speeding up the process of creating and tuning of knowledge bases and expert system deployment. Currently, the NPS32 knowledge representation is implemented and has been tested and verified. Another method used for speeding up the tuning process was proposed (chapter 3 of this paper) and will now be implemented into the shell and tested for its suitability for industrial (in-process) use.

References: