Available online at www.sciencedirect.com
SCIENCE DIRECT®
Data & Knowledge Engineering 55 (2005) 103–128
DATA & KNOWLEDGE ENGINEERING
www.elsevier.com/locate/datak

Rule-based update methods for a hybrid rule base

Jim Prentzas [b,c], Ioannis Hatzilygeroudis [a,b,*]

[a] University of Patras, School of Engineering, Department of Computer Engineering and Informatics, 26500 Patras, Greece
[b] Research Academic Computer Technology Institute, P.O. Box 1122, 26110 Patras, Greece
[c] Technological Educational Institute of Lamia, Department of Informatics and Computer Technology, 35100 Lamia, Greece

Available online 10 March 2005

Abstract

In this paper, we present methods for efficient updates of a hybrid rule base. The hybrid rule base consists of neurules, a type of hybrid rules combining symbolic rules and neural networks. A neurule base, called the target knowledge, is produced by conversion from a symbolic rule base, called its source knowledge. The presented methods concern modifications to the target knowledge, due to insertion of a new rule in or removal of an old rule from its source knowledge. The methods (a) require as little re-conversion as possible and (b) preserve the number of neurules as small as possible. This is achieved by storing information related to the conversion process in a tree, called the splitting tree. Experimental results demonstrate the benefits of using the splitting tree.

Keywords: Hybrid rule bases; Rule base maintenance; Rule insertion methods; Rule deletion methods

1. Introduction

There has been extensive research activity at combining (or integrating) the symbolic and the connectionist approaches for problem solving in intelligent systems [3,7,12,14,15,21,24].

* Corresponding author. Tel.: +30 2610996937; fax: +30 2610960374.
E-mail addresses: dprentzas@teilam.gr (J. Prentzas), ihatz@cti.gr, ihatz@ceid.upatras.gr (I. Hatzilygeroudis).

0169-023X/$ - see front matter
doi:10.1016/j.datak.2005.02.001

Especially, there are a number of efforts combining symbolic rules and neural networks [4–6,8,13,16,20,22,23]. The main objective is to reduce knowledge elicitation from experts to a minimum. A common characteristic of them is that they give pre-eminence to connectionism, so they use a neural network as their knowledge base. Connectionism is mainly used as a means for refining an initial background rule-base. A weak point of them is that their knowledge base lacks the naturalness and modularity of symbolic rules; it is incomprehensible. So, explanations are often provided in the form of if–then rules by rule extraction methods [1,17].

Neurules integrate symbolic rules and connectionism, but in a different way. They give pre-eminence to the symbolic component [11]. Neurocomputing is used within the symbolic framework to improve the performance of symbolic rules [9]. A symbolic rule base is converted into a hybrid one, a neurule base. The neurule base is proved to be quite smaller than the initial rule base, since in average each neurule is a merger of more than one symbolic rule, retains its modularity, since it consists of autonomous units (neurules), and also retains its naturalness in a great degree, since neurules look much like symbolic rules [10]. Also, the corresponding inference mechanism, which is a tightly integrated process, results in more efficient inferences than those of symbolic rules, and explanations, in the form of if–then rules, can be provided [11].

So, the goal of neurules is not knowledge refinement, but gain in space and time efficiency in rule-based reasoning, without sacrificing much of naturalness and modularity. One consequence of modularity in symbolic rule-based expert systems is the ability of incremental development of the knowledge base. Furthermore, incremental maintenance of a rule base is possible too: one can easily update, e.g. insert a rule in or remove a rule from a rule base. To preserve those advantages for neurule bases as well, we need to find methods for easy updating. Thus, given that the initial rule base may change, due to updates, the problem of maintaining the produced neurule base to reflect those changes, without performing extended re-conversion, arises.

In this paper, which is an extension of [18], we present methods for efficient updates of the knowledge base (target knowledge) of a neurule-based expert system, due to changes to its source knowledge. Section 2 presents neurules. Section 3 introduces the methods, presents some examples and discusses the correctness of the methods. Section 4 presents experimental results comparing the update methods. Finally, Section 5 concludes.

2. Neurules

2.1. Syntax and semantics

Neurules are a kind of hybrid rules. The form of a neurule is depicted in Fig. 1a. Each condition C_i is assigned a number sf_i, called its *significance factor*. Moreover, each rule itself is assigned a number sf_0, called its *bias factor*. Internally, each neurule is considered as an adaline unit (Fig. 1b). The *inputs* C_i ($i = 1,\ldots,n$) of the unit are the (values of the) *conditions* of the rule. The weights of the unit are the significance factors of the neurule and its bias is the bias factor of the neurule. Each input takes a value from the following set of discrete values: [1 (true), 0 (false), 0.5 (unknown)]. The *output D*, which represents the *conclusion* (decision) of the rule, is calculated via the standard formulas

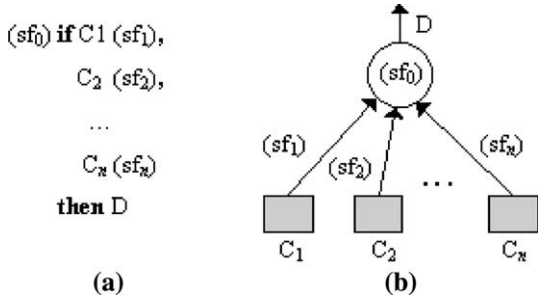

Fig. 1. (a) Form of a neurule and (b) a neurule as an adaline unit.

$$D = f(a), a = sf_0 + \sum_{i=1}^{x} sf_i C_i$$

$$f(a) = \begin{cases} 1 & \text{if } a \geqslant 0 \\ -1 & \text{otherwise} \end{cases}$$

where a is the *activation value* and $f(x)$ the *activation function*, a threshold function. Hence, the output can take one of two values ('−1','1') representing failure and success of the rule, respectively.

The general syntax of a condition C_i and the conclusion D is:

<condition> ::= <variable> <l-predicate> <value>
<conclusion> ::= <variable> <r-predicate> <value>

where <variable> denotes a variable, that is a symbol representing a concept in the domain, e.g. 'sex', 'pain', etc., in a medical domain. <l-predicate> and <r-predicate> are one of {is, isnot}. <value> denotes a value. It can be a symbol or a number. The significance factor of a condition represents the significance (weight) of the condition in drawing the conclusion. The LMS learning algorithm (see e.g. in [6]) is used to compute the values of the significance factors as well as the bias factor of a neurule. So, the semantics of the significance factors is different from that of certainty factors [2].

Corresponding *symbolic rules* have the same syntax as that in Fig. 1a, but without any factors and by ',' denoting conjunction (see Table 1 for examples of a symbolic rule and a neurule).

Table 1
A symbolic rule (R1) and a neurule (N1)

R1	N1
if patient-class is human0–20, pain-feature2 is continuous, fever is no-fever, antinflam-reaction is medium **then** disease is arthritis	(−5.6) **if** antinflam-reaction is medium (8.7), patient-class is human21–35 (8.7), pain-feature2 is night (8.5), pain-feature2 is continuous (5.1), fever is no-fever (5.0), patient-class is human0–20 (4.6) **then** disease is arthritis

2.2. Construction of a Neurule-base

2.2.1. Basic conversion algorithm

A neurule base (NRB) can be constructed by conversion from a symbolic rule base (SRB). Existing (propositional type) symbolic rule bases can be easily transformed into a symbolic rule base of the above syntax and then converted to a neurule base. A symbolic rule base may also be the result of direct knowledge elicitation from experts or the product of an automated knowledge acquisition method (e.g. ID3 or C4.5 algorithms). The conversion of an SRB to an NRB is achieved by applying the *basic conversion algorithm* (BCA) [9]. Application of BCA does not result in a refinement of the converted symbolic rule-base, but creates an equivalent knowledge base. This means that the conclusions drawn from NRB are the same as those drawn from SRB, given the same inputs. SRB is called the *source knowledge*, whereas NRB is called the *target knowledge* (see Fig. 2).

BCA tries to merge all symbolic rules having the same conclusion into one neurule. However, this is not always possible, due to non-linearity problems, as it is explained later in this section. In any case, each produced neurule usually merges two or more symbolic rules with the same conclusion. In this way, the size of the produced NRB (target knowledge) is less than that of SRB (source knowledge), as far as both the number of rules and the number of conditions are concerned.

BCA is outlined as follows:

1. Group symbolic rules into *merger sets*.
2. From each merger set, produce a *merger*.
3. Produce a training set for each merger.
4. Train each merger and produce one or more neurules.

Each *merger set* contains all the rules of the SRB having the same conclusion. A *merger* is a neurule having as conditions all the conditions of the symbolic rules in the corresponding merger set (without duplications) and significance factors as well as bias factor set to zero (or any other proper initial value). Each training set is extracted from the truth table of the combined logical function of the rules in its merger set (the disjunction of the conjunctions of the conditions of each rule), via a filtering process. Filtering eliminates the invalid rows of the truth table. Invalid rows are those with contradicting or inconsistent values, e.g. those having 'true' ('1') at both 'fever is high' and 'fever is low' columns. So, finally, the training set contains all acceptable *training patterns/rows* (for a detailed treatment see [9]).

Training of mergers is performed using the standard LMS algorithm (see e.g. in [6]). Training, however, is not always successful; it cannot always find a set of significance and bias factors that

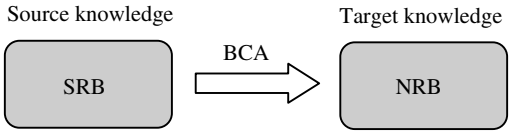

Fig. 2. Neurule base production.

classify correctly all of the training patterns. Training of a merger is not successful in case that its training set corresponds to a non-linear (boolean) function or, in other words, the training patterns of the training set are *inseparable*. When the algorithm succeeds, that is values for the significance factors are calculated that successfully classify all training patterns, one neurule is produced. When it fails (case of inseparable training patterns), a splitting process is followed, which produces more than one neurule having the same conclusion, called *sibling neurules*. To this end, the notion of *closeness* between two symbolic rules is used to guide splitting. The closeness between two symbolic rules is defined as the number of their common conditions. A *least closeness pair* (*LCP*) of rules in a merger set is a pair of rules with the *least closeness* (*LC*) in the set. There may be more than one LCP in a merger set.

So, in case of inseparability, the initial merger set is divided into subsets in a way that each subset contains symbolic rules that are "close" to each other in some degree. Initially, a LCP in the merger set is found and two subsets are created each containing as its initial element one of the rules of that pair, called its *pivot*. Each of the remaining rules is distributed between the two subsets based on their closeness to their pivots. That is, each subset contains rules, which are closer to its pivot. If training fails, for a merger of a merger subset, the corresponding subset is further split into two other subsets, based on one of its LCPs. This continues, until training succeeds or the merger subset contains only one rule called the *remaining rule*. This kind of splitting stems from the observation that separable sets have rules with larger average closeness than inseparable ones.

Thus, step 4 of the conversion algorithm is analyzed as follows:

4.1. Train the merger using the specified training set.
4.2. If training fails, find an LCP and produce two subsets of the merger set, each having as initial element one of the LCP rules respectively, called its pivot. In each merger subset put the symbolic rules (of the initial merger set), which are closer to its pivot.
4.3. For each merger subset, apply step 4.1 recursively until either training succeeds or the merger subset contains only one rule (remaining rule).
4.4. Convert any remaining rule into a neurule.

As an example, to demonstrate application of the main steps of BCA, we use the merger set of Table 2 that consists of five symbolic rules $\{R1, R2, R3, R4, R5\}$, taken from a medical diagnosis rule base. The merger constructed from this merger set contains the nine distinct conditions of the five rules and is shown in Table 3. The training set of the merger is extracted from the truth table of the combined logical function of the rules of the merger set:

$$F = (C1 \wedge C2 \wedge C3 \wedge C4) \vee (C1 \wedge C5 \wedge C3 \wedge C4) \vee (C6 \wedge C5 \wedge C4) \vee (C7 \wedge C2 \wedge C8 \wedge C9) \vee (C6 \wedge C2 \wedge C3 \wedge C4)$$

where C1 ≡ patient-class is human0–20, C2 ≡ pain-feature2 is continuous, C3 ≡ fever is no-fever, C4 ≡ antinflam-reaction is medium, C5 ≡ pain-feature2 is night, C6 ≡ patient-class is human21–35, C7 ≡ patient-class is human36–55, C8 ≡ joints-pain is no, C9 ≡ antinflam-reaction is high.

The truth table of F contains $2^9 = 512$ training patterns, from which only 144 patterns remain after application of the filtering process. Due to non-linearity, training of the merger is not successful and the initial merger set is split in two subsets: $MS_1 = \{R1, R2, R3, R5\}$ and $MS_2 = \{R4\}$.

Table 2
An example merger set of symbolic rules

R1
if patient-class is human0–20,
pain-feature2 is continuous,
fever is no-fever,
antinflam-reaction is medium
then disease is arthritis

R2
if patient-class is human0–20,
pain-feature2 is night,
fever is no-fever,
antinflam-reaction is medium
then disease is arthritis

R3
if patient-class is human21–35,
pain-feature2 is night,
antinflam-reaction is medium
then disease is arthritis

R4
if patient-class is human36–55,
pain-feature2 is continuous,
joints-pain is no,
antinflam-reaction is high
then disease is arthritis

R5
if patient-class is human21–35,
pain-feature2 is continuous,
fever is no-fever,
antinflam-reaction is medium
then disease is arthritis

Table 3
The merger of the merger set of Table 2

(0) **if** patient-class is human0–20 (0),
pain-feature2 is continuous (0),
fever is no-fever (0),
antinflam-reaction is medium (0),
pain-feature2 is night (0),
patient-class is human21–35 (0),
patient-class is human36–55 (0),
joints-pain is no (0),
antinflam-reaction is high (0)
then disease is arthritis

The rules of the first merger subset are "close" to each other due to their common conditions, whereas R4 has few common conditions with the other rules. The LCP that guides splitting is (R2, R4).

The merger of the first subset is successfully trained and the corresponding neurule is produced (NR1-2-3-5, Table 4). The second merger subset contains only R4 (remaining rule), which is converted into a neurule, using as training set the patterns from its truth table (NR4, Table 4). Both neurules are inserted into the neurule-base. So, finally, from the initial merger set of five symbolic rules, two neurules are produced.

As indicated above, the size of target knowledge (NRB) is less than that of source knowledge (SRB). This is true in terms of both, the number of rules and the number of conditions, and results

Table 4
Neurules produced from the merger set of Table 2

NR1-2-3-5	NR4
(−5.6) **if** antinflam-reaction is medium (8.7), patient-class is human21–35 (8.7), pain-feature2 is night (8.5), pain-feature2 is continuous (5.1), fever is no-fever (5.0), patient-class is human0–20 (4.6) **then** disease is arthritis	(−7.4) **if** antinflam-reaction is high (3.2), patient-class is human36–55 (2.9), joints-pain is no (2.6) pain-feature2 is continuous (2.6) **then** disease is arthritis

in improvements to the efficiency of the inferences from NRB, compared to those from SRB. More specifically, the decrease in the size of the rule base leads to the following advantages:

(i) Less computations are required for the evaluation of conditions.
(ii) Smaller conflict sets are formed when pursuing intermediate or final goals, due to the fact that the number of rules with the same conclusion is reduced.

These factors have a direct impact on the inference efficiency. In [9] we provide experiments comparing the inference efficiency of source and target knowledge.

Table 5
An example merger set of symbolic rules

R1	R2
if arterial-conc is slight-incr, blood-conc is normal, scan-conc is normal, capill-conc is mod-incr, venous-conc is highly-incr **then** disease-type is early-inflammation	**if** arterial-conc is mod-incr, blood-conc is highly-incr, scan-conc is normal, capill-conc is slight-incr, venous-conc is slight-incr **then** disease-type is early-inflammation
R3	R4
if arterial-conc is mod-incr, blood-conc is normal, scan-conc is normal, capill-conc is slight-incr, venous-conc is normal **then** disease-type is early-inflammation	**if** arterial-conc is mod-incr, blood-conc is mod-incr, scan-conc is normal, capill-conc is mod-incr, venous-conc is slight-incr **then** disease-type is early-inflammation
R5	R6
if arterial-conc is mod-incr, blood-conc is normal, scan-conc is normal, capill-conc is mod-incr, venous-conc is slight-incr **then** disease-type is early-inflammation	**if** arterial-conc is mod-incr, blood-conc is slight-incr, scan-conc is normal, capill-conc is mod-incr, venous-conc is mod-incr **then** disease-type is early-inflammation

The conversion algorithm presented in [9] did not transduce single symbolic rules, resulting in a target knowledge consisting of both neurules and symbolic rules. In the conversion algorithm outlined above, single symbolic rules are transduced into neurules. This is done in order to obtain a homogeneous target knowledge, consisting only of neurules. This also results in a less complicated inference process and plays a positive role in the efficiency of the inference engine.

2.2.2. Splitting tree

For reasons that will become clear in the next section, the splitting process for each initial merger set is stored as a tree, which is called its *splitting tree*. The root of the tree corresponds to the initial merger set. The intermediate nodes and leaves correspond to the subsequent subsets, into which the initial merger set was split. An intermediate node denotes a subset that was split, due to training failure, whereas a leaf denotes a subset that was successfully trained and produced a neurule. The pivot of each (sub)set is attached to the corresponding branch of the tree. It can be easily seen that the merger (sub)set of the root or an intermediate node is a superset of the merger subsets related to its descendant nodes. Furthermore, the nearer one gets to the leaves, the greater the mean closeness between the rules of the corresponding merger subsets. Tree information is stored in NRB alongside the produced neurules.

To illustrate splitting and its representation, we use the merger set of rules R1–R6, presented in Table 5, from which five neurules are produced, shown in Table 6. Fig. 3 depicts the corre-

Table 6
Neurules produced from the merger set of Table 5

NR1
(−11.4) **if** venous-conc is high-incr (3.3),
arterial-conc is slight-incr (3.0),
blood-conc is normal (2.8),
scan-conc is normal (2.7),
capill-conc is mod-incr (2.7)
then disease-type is early-inflammation

NR2
(−11.4) **if** venous-conc is slight-incr (3.3),
arterial-conc is mod-incr (3.0),
blood-conc is high-incr (2.8),
scan-conc is normal (2.7),
capill-conc is slight-incr (2.7)
then disease-type is early-inflammation

NR3
(−11.4) **if** venous-conc is normal (3.3),
arterial-conc is mod-incr (3.0),
blood-conc is normal (2.8),
scan-conc is normal (2.7),
capill-conc is slight-incr (2.7)
then disease-type is early-inflammation

NR4–5
(−7.8) **if** venous-conc is slight-incr (3.3),
arterial-conc is mod-incr (3.0),
blood-conc is mod-incr (2.8),
scan-conc is normal (2.7),
capill-conc is mod-incr (2.7),
blood-conc is normal (2.6),
then disease-type is early-inflammation

NR6
(−11.4) **if** venous-conc is mod-incr (3.3),
arterial-conc is mod-incr (3.0),
blood-conc is slight-incr (2.8),
scan-conc is normal (2.7),
capill-conc is mod-incr (2.7)
then disease-type is early-inflammation

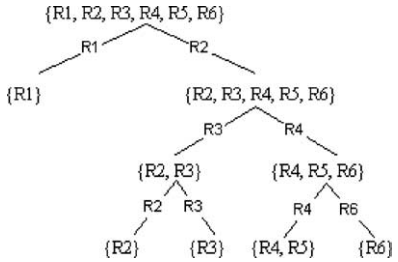

Fig. 3. The splitting tree for the merger set of Table 3.

sponding splitting tree. Due to non-linearity, the initial merger set $\{R1, R2, R3, R4, R5, R6\}$ is split in two subsets: $\{R1\}$ and $\{R2, R3, R4, R5, R6\}$, with LCP: $(R1, R2)$ (LC = 1). $\{R1\}$ produces neurule NR1. $\{R2, R3, R4, R5, R6\}$ is split in $\{R2, R3\}$ and $\{R4, R5, R6\}$, with LCP: $(R3, R4)$ (LC = 2). Then, $\{R2, R3\}$ is split in $\{R2\}$ and $\{R3\}$, so NR2 and NR3 are produced and so on.

3. Update methods-algorithms

3.1. Update requirements

As knowledge evolves, however, the source knowledge (SRB) may be updated. This implies that the target knowledge (NRB) should be also updated, to reflect those changes.

In Fig. 4, the architecture of the maintenance system of a neurule-based system is illustrated. UM is the *update mechanism*, which implements update methods-algorithms. CM is the *conversion mechanism*, which basically implements BCA. Finally, R is a symbolic rule, which represents the rule involved in the update.

The basic changes to SRB can be (a) *insertion* of a new rule and (b) *removal* (or *deletion*) of an existing rule, since modification of a rule is equivalent to removal of the old rule and insertion of the new one. In either case, the straightforward way is to re-produce NRB. However, re-production of the whole NRB implies a large computational effort. On the other hand, re-production involves useless re-conversions, i.e. re-conversions of non-affected large portions of SRB.

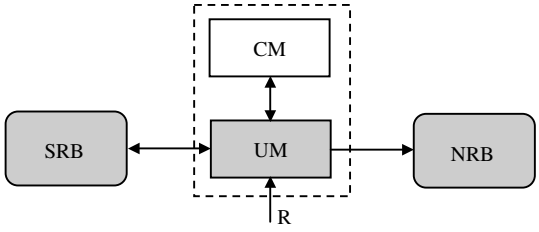

Fig. 4. Architecture of the maintenance system of a neurule-based system.

Therefore, methods for updating the target knowledge without re-conversion of the whole SRB should be developed. These methods should be efficient in two ways:

(i) They should require the least possible computational effort or, in other words, re-convert as small portion of source knowledge as possible.
(ii) The number of neurules after an update should remain as small as possible.

The second requirement plays an important role in the efficiency of the neurule-based inference process. The number of neurules in NRB affects the number of computations required for the evaluation of the rule conditions [9]. Furthermore, it is desirable to keep the number of neurules with the same conclusion as small as possible, so that the size of the conflict set during inference will be as small as possible too.

Due to the modularity of NRB, inserting a new rule into or removing an existing rule from the source knowledge does not affect the entire NRB (target knowledge), but only the neurules produced from the initial merger set related to that rule. Furthermore, not all corresponding neurules are affected, but only those produced from specific merger subsets of the initial merger set. The rest of NRB remains intact. In the sequel, we introduce update methods dealing with rule insertion and removal in an efficient way.

3.2. Rule insertion

Given a new symbolic rule R, to be inserted in SRB, three cases can be distinguished:

(a) There is no sibling neurule of R in NRB.
(b) There is only one sibling neurule of R in NRB.
(c) There are more than one sibling neurule of R in NRB.

Case (a) is the simplest. The new symbolic rule is converted into a neurule and inserted into NRB.

If (b) is the case, it means that the corresponding merger set was not split. To handle this case, the existing sibling neurule is removed from NRB and a new merger set is formed containing the new symbolic rule and the symbolic rules of the initial merger set. The new merger is formed and trained. If training is successful, one neurule is produced. If training fails, two neurules are produced.

Case (c) is the most difficult. The existence of more than one neurule with the same conclusion means that, due to inseparability, the initial merger set was split into a number of merger subsets. There can be three approaches to handle this case:

(i) Merely, convert the new symbolic rule into a neurule and insert it into NRB. This method, called the *simple* (*SI*) *method*, is computationally the most efficient, but definitely increases the number of neurules in NRB, which is not desirable. This increase happens because the method does not check whether the new symbolic rule can be merged with other existing symbolic rules. Despite its disadvantage, this method demonstrates the modularity of neurules; it is similar to the way new symbolic rules are inserted into a symbolic rule base, given

that they do not contradict with any of the existing rules. This method could also have been used to handle case (b), but, due to the fact that only one neurule was produced from the initial merger set (prior the insertion of the new rule), the computational benefit is insignificant.

(ii) The existing neurules are removed from NRB, the new symbolic rule is merged with the initial merger set and training of the new merger is performed to produce the revised neurules. This approach, called the *total retraining* (*TR*) *method*, is inefficient as far as the computational effort is concerned, especially when more than two neurules are produced from the initial merger set. This happens, because the information contained in the splitting tree is not taken into account, thus performing extra training and splitting.

(iii) The third approach, called the *splitting tree* (*ST*) *method*, exploits the information contained in the splitting tree so that the update process focuses on the neurules produced from the merger subset containing the rules closest to R. In this way, the least possible subset of neurules produced from the initial merger set is affected. This is achieved by traversing the splitting tree, starting from the root. Traversing is based on the closeness of the inserted rule to the LCP members of the merger subsets corresponding to the traversed nodes. Traversing ends at an intermediate node, when the corresponding merger subset contains a rule whose closeness to the inserted rule is less than LC. Otherwise, traversing ends at a leaf.

More specifically, the algorithm of the ST approach is as follows:

Starting from the root, traverse the splitting tree

1. If the current node is not a leaf, check whether the merger (sub)set corresponding to the node contains a rule R′ whose closeness to R is less than LC of the (sub)set. If there is no such rule, insert R into the merger (sub)set of the node and execute this step recursively for the child of the node denoted by the LCP member closer to R. If there is such a rule R′, do:
   1.1. Stop traversing the splitting tree.
   1.2. Remove from the splitting tree all nodes descending from current node and from NRB all neurules corresponding to the leaves descending from current node. The removed nodes and neurules are inserted into a temporary buffer.
   1.3. Insert R into the corresponding merger (sub)set and split it into two subsets with LCP: (R, R′).
   1.4. Train each one of the mergers formed from the two subsets, produce the corresponding neurules (reusing parts of the initial splitting tree contained in the buffer to avoid unnecessary training or splitting), insert the produced neurules into NRB and update the splitting tree. Moreover, clear the temporary buffer.
2. If the current node is a leaf, remove the corresponding neurule, insert R into its merger set and train the merger. If training fails, split the merger set, produce the two neurules, insert them into NRB and update the splitting tree. If training is successful, do the following:
   2.1. If the sibling node of the leaf is also a leaf and introduction of R into their parent's merger subset increases the mean closeness between its rules, and the parent node's new merger can be successfully trained, do the following:
      2.1.1. Remove the neurule corresponding to the sibling leaf from NRB.

2.1.2. Insert the neurule produced from the parent node's new merger into NRB.
2.1.3. Update the splitting tree.
2.2. Else do the following:
2.2.1. Insert the neurule produced from the leaf's new merger into NRB.
2.2.2. Update the splitting tree.

As can be easily seen from the description of the third approach, most of the changes are required when traversing stops at an intermediate node. In that case, the new merger set corresponding to the intermediate node has to be split again, due to the fact that LC has been decreased compared to the merger set prior to the introduction of the new rule. Splitting is based on the new LCP.

In step 2.1, the mean closeness of the parent merger set is checked because, if it is increased, it is likely that the corresponding new merger can be successfully trained. In case of successful training of the new merger of the father, the number of neurules contained in the neurule base decreases by one (steps 2.1.1–2.1.3).

As an example, consider the rules in Table 5 as constituting SRB and those in Table 6 as constituting NRB. Also, suppose that rule R7 (Table 7) is to be inserted. Given that more than one neurule have the same conclusion as R7, it is a (c) case. Following the ST method, traversing the tree ends at the leaf related to subset {R2} (Fig. 5a). Notice that R7 is inserted into the merger sets corresponding to all traversed nodes. Training of the new merger (sub)set {R2, R7} is successful. Thus, NR2 is removed from NRB. The sibling node of {R2} is also a leaf (Fig. 5b) and insertion of R7 into subset {R2, R3}, related to their 'parent' node (Fig. 6a), increases its mean closeness from 3 ({R2, R3}) to 11/3 ({R2, R3, R7}). Therefore, training of the merger of {R2, R3, R7} is tried. It is successful and NR2-3-7 is produced and inserted into NRB. Meanwhile, NR3 is removed from NRB. The splitting tree takes the form in Fig. 6b. So, insertion of R7 finally decreases the total number of neurules in NRB from five to four (NR1, NR2-3-7, NR4-5, NR6).

Notice that ST method focuses on subset {R2, R3}, which includes the rules closest to R7. Thus, only the necessary part of NRB is (re)trained. The part of NRB produced from R1, R4, R5 and R6 remains intact. TR method would have produced the same neurules, requiring though unnecessary training and splitting. SI method would have inserted the neurule produced from R7 into NRB. The computational effort for the update would have been minimal. However, at the end, NRB would contain six neurules instead of the four resulted by ST method.

Table 7
Inserted rule R7 and resulted neurule NR2-3-7

R7	NR2-3-7
if arterial-conc is mod-incr, blood-conc is normal, scan-conc is normal, capill-conc is slight-incr, venous-conc is slight-incr **then** disease-type is early-inflammation	(−20.4) **if** venous-conc is slight-incr (8.7), arterial-conc is mod-incr (8.4), scan-conc is normal (8.1), capill-conc is slight-incr (8.1), blood-conc is normal (8.0), blood-conc is highly-incr (4.6) venous-conc is normal (1.5) **then** disease-type is early-inflammation

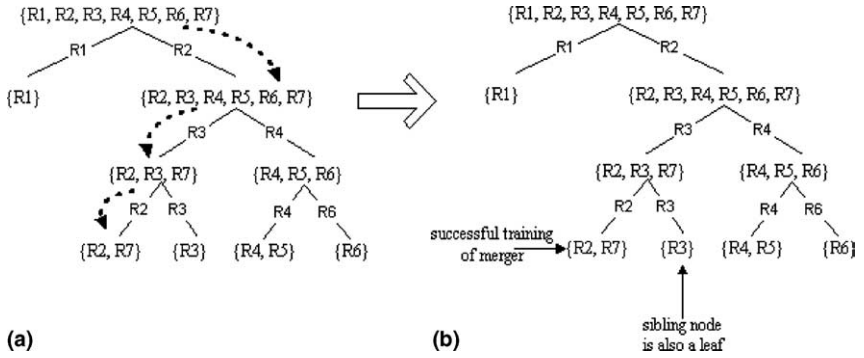

Fig. 5. Insertion of R7 (I) (a) traversal of the splitting tree and (b) checking the leaf node at which traversing stopped.

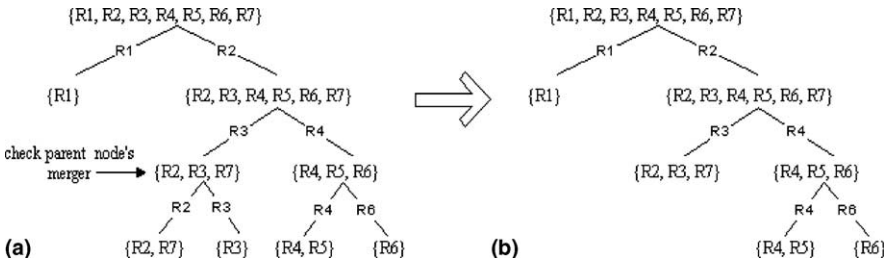

Fig. 6. Insertion of R7 (II) (a) checking parent node's merger and (b) final form of splitting tree.

3.3. Rule removal

Given the need to remove (or delete) a symbolic rule R from SRB, two cases can be distinguished:

(a) There is only one sibling neurule of R in NRB.
(b) There are more than one sibling neurule of R in NRB.

If (a) is the case, it means that the corresponding merger set was not split. So, the corresponding neurule is removed from NRB. If the merger set includes only the removed rule, nothing else is done. Otherwise, a new merger set is formed, including the rules of the initial merger set, but excluding the removed rule. The new merger is trained. If training is successful, one neurule is produced. If it fails, the merger set is split and, after training, two neurules are produced.

There can be three approaches to handle case (b):

(i) In this approach, called the *simple* (*SI*) *method*, only the neurule whose merger set includes the removed rule is affected. The neurule is removed from NRB. If the merger set includes only the rule that is removed, nothing else is done. Otherwise, the new merger (formed as in (a)) is trained and corresponding neurule(s) is(are) produced. This method is computationally efficient, but may increase the number of sibling neurules.

(ii) All the neurules derived from the initial merger set are removed from NRB and the merger of the initial merger set, after excluding the removed rule, is trained. After possible splitting, the corresponding neurule(s) is(are) produced. This approach, which is called the *total retraining* (*TR*) *method*, is inefficient, because by discarding the information contained in the splitting tree it performs training and splitting that could have been avoided.

(iii) The third approach, called the *splitting tree* (*ST*) *method*, exploits the information contained in the splitting tree in a way similar to approach (iii) for rule insertion. The update process thus focuses on the part of the NRB produced from the merger subset with the rules closest to the removed one. More specifically, the third approach is as follows.

Starting from the root, traverse the splitting tree:

1. If the current node is not a leaf, check whether R is a member of LCP of its merger (sub)set. If it is not, remove R from the merger (sub)set and execute this step recursively for the child of the node on the edge denoted by the LCP's member closer to R. If it is, do:
   1.1. Stop traversing the splitting tree.
   1.2. Remove from the splitting tree all nodes descending from this node and from NRB all neurules produced from this merger (sub)set. The removed nodes and neurules are inserted into a temporary buffer.
   1.3. Remove R from the node's merger (sub)set and train the resulted merger. If training fails, produce the neurules (after splitting), insert them into NRB, update the splitting tree (possibly reusing parts of the initial splitting tree) and clear the temporary buffer. If training is successful, do the following:
      1.3.1. If the sibling node of the leaf is also a leaf and removal of R from their parent's merger subset increases the mean closeness between its rules and the parent's new merger can be successfully trained, do the following:
         1.3.1.1. Remove the neurule corresponding to the sibling leaf from NRB.
         1.3.1.2. Insert the neurule produced from the parent node's new merger into NRB.
         1.3.1.3. Update the splitting tree and clear the temporary buffer.
      1.3.2. Else do the following:
         1.3.2.1. Insert the neurule produced from the leaf's new merger into NRB.
         1.3.2.2. Update the splitting tree and clear the temporary buffer.
2. If the current node is a leaf, do:
   2.1. Remove the neurule whose merger set included R from NRB.
   2.2. Remove R from the leaf's merger set.
   2.3. Form the new merger and train it.
   2.4. If training fails, split the merger set, produce the two neurules, insert them into NRB and update the splitting tree.
   2.5. If training succeeds, do:
      2.5.1. If the sibling of the leaf node is also a leaf and removal of R from the merger set of their parent increases its mean closeness and the parent's new merger can be successfully trained, do:

2.5.1.1. Remove the neurule corresponding to the sibling leaf from NRB.
2.5.1.2. Insert the neurule produced from the parent's new merger into NRB.
2.5.1.3. Update the splitting tree.
2.5.2. Else do:
2.5.2.1. Insert the neurule produced from the leaf's new merger into NRB.
2.5.2.2. Update the splitting tree.

As a first example, suppose that R5 is to be removed (after R7 insertion, Fig. 6b). It is a (b) case and following the ST method, since R5 is not a member of any LCP, traversing ends at the leaf related to subset {R4, R5}. Notice that R5 is removed from the merger subsets corresponding to the traversed nodes (Fig. 7a). NR4-5 is removed from NRB, the merger of {R4} is trained and NR4 is produced. The sibling node of leaf {R4} is also a leaf corresponding to subset {R6} (Fig. 7b). Removal of R5 decreases the mean closeness of their parent node's subset from 10/3 ({R4, R5, R6}) to 3 ({R4, R6}). Therefore, no training of the merger of {R4, R6} is tried. NR4 is inserted into NRB and the splitting tree is updated (Fig. 8b). Once again, a large portion of NRB remains intact. The new NRB consists of NR1, NR2-3-7, NR4 and NR6.

As a second example, suppose that R6 is to be removed (after R7 insertion). R6 is not member of any LCP as far as node {R4, R5, R6}, so R6 is removed from the subsets of its ancestor nodes and also from {R4, R5, R6} (Fig. 9a). Neurules NR4-5 and NR6 as well as the nodes corresponding to merger subsets {R4, R5} and {R6} are removed from NRB and the splitting tree respectively and are inserted into the (temporary) buffer. The merger of the new subset {R4, R5} of the node in which the splitting tree traversal stopped (Fig. 9b) can be trained successfully based on the information contained in the temporary buffer. The sibling node of this node is also a leaf (corresponding to subset {R2, R3, R7}) and removal of R6 from the merger subset of their parent node increases its mean closeness from 3.07 ({R2, R3, R4, R6, R7}) to 3.4 ({R2, R3, R4, R7}). Therefore, training of the merger of subset {R2, R3, R4, R7} is tried (Fig. 10a). Training is not successful and thus neurule NR4-5 is inserted into NRB (reusing information contained in the buffer). The splitting tree is updated (Fig. 10b) and the buffer is cleared. Notice that NR4-5 needs no reproduction, it is simply reused.

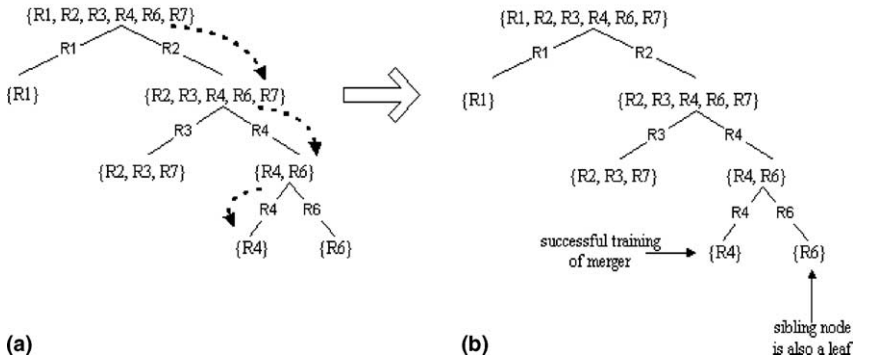

Fig. 7. Removal of R5 (I) (a) traversal of the splitting tree and (b) checking the leaf node at which traversing stopped.

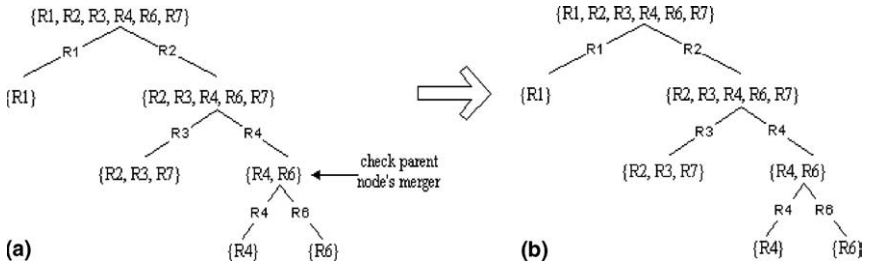

Fig. 8. Removal of R5 (II) (a) checking parent node's merger and (b) final form of splitting tree.

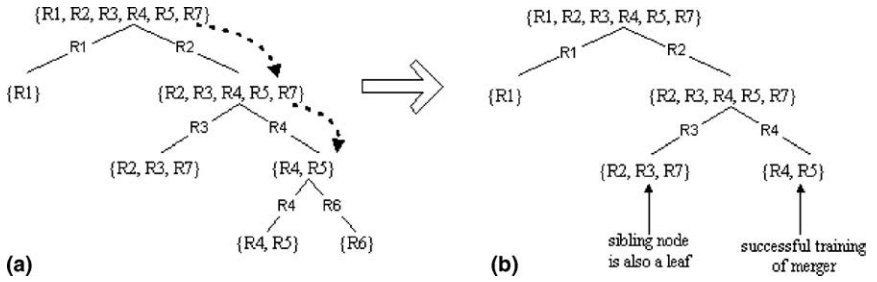

Fig. 9. Removal of R6 (I) (a) traversal of the splitting tree and (b) checking the node at which traversing stopped.

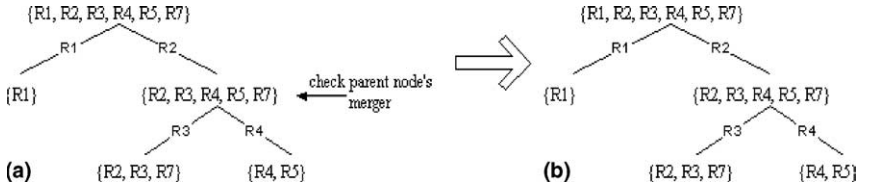

Fig. 10. Removal of R6 (II) (a) checking parent node's merger and (b) final form of splitting tree.

3.4. About correctness

Given that the above algorithms concern indirect changes to target knowledge, the question about their correctness is reasonably set. The basis of the algorithms is BCA. The criterion for the correctness of that algorithm is to preserve inference equivalence between SRB and NRB, which means that all inferences performed from SRB should also be performed from NRB (with identical results). This is implicitly proved and mainly experimentally confirmed in [9]. The proof is based on the fact that the training set of each merger includes the (valid) rows of the truth table of the combined logical function of the rules in the corresponding merger set. This means that the truth table represents all (valid) conclusions that can be drawn from the rules. Given that training is complete, that is all training patterns should be successfully classified, inference equivalence is assured.

Inserting a new rule in SRB means that a new inference is introduced. So, the rule insertion update algorithms of Section 3.2, to be correct, should assure that the new inference can also be performed in the updated NRB. This is rather obvious, since in any case the truth table of the new inserted rule is taken into account in creating the new training set for the partial reconversion. A similar thing happens when a rule is removed from SRB. The patterns related to its truth table are removed from the training set of the corresponding merger. So, correctness of the rule removal update algorithms of Section 3.3 is also assured.

4. Experiments and discussion

4.1. Introductory aspects

This section presents experimental results related to the update methods. The experimental results were produced by using two medical symbolic rule bases. Table 8 shows the (common) conclusions of the rules of each merger set that was produced. Table 9 contains the general characteristics of the produced merger sets. For each merger set, it shows the total number of the merged symbolic rules, the number of distinct conditions in the corresponding merger and the number of neurules produced from the initial merger set. Section 4.2 presents experimental results for comparing the rule insertion update methods, whereas section 4.3 presents experimental results for comparing the rule removal update methods.

The comparison between the update methods is based on two measures, which stem from the requirements set in Section 3.1. The first is the total number of produced neurules (directly stemming from the first requirement). The second is the number of mergers (of the merger subsets) that had to be (re)trained, because of the changes to the source knowledge, (stemming from the second requirement). Indeed, the number of required (re)trainings is a measure of the required (re)conversion effort, because, in the algorithms presented in Section 3, training a merger is the far most computationally expensive process, thus has the most significant impact on the required run time. Although the number of required trainings is an adequate measure, to increase the certainty of

Table 8
Merger sets and corresponding conclusions

Merger set	Conclusion
M1	Disease-type is inflammation
M2	Disease-type is arthritis
M3	Disease-type is primary-malignant
M4	Disease-type is special-arthritis
M5	Disease-type is secondary-malignant
M6	Disease-type is early-inflammation
M7	Disease-type is soft-tissue-inflammation
M8	Disease-type is soft-tissue-early-bone-inflammation
M9	Disease-type is soft-tissue-bone-inflammation
M10	Disease-type is early-soft-tissue-inflammation
M11	Disease-type is bone-inflammation-or-benign-tumor
M12	Disease-type is intensive-soft-tissue-inflammation

Table 9
General characteristics of the merger sets

Merger set	Total merged symbolic rules	Number of distinct conditions	Number of produced neurules
M1	8	9	3
M2	5	9	2
M3	10	10	2
M4	4	10	3
M5	3	7	2
M6	9	13	5
M7	20	14	6
M8	7	13	3
M9	4	9	2
M10	7	13	4
M11	4	10	2
M12	5	11	2

our conclusions, we also made runtime measurements in our experiments. It is fact that not all merger trainings are equally expensive, because they depend on the size of the training sets of the mergers. Merger subsets higher up in the splitting tree are bigger in size than those lower down, hence training of the former is more expensive than that of the latter. This cannot be captured by using as a measure only the number of trained mergers.

Furthermore, our experiments correspond to a number of possible scenarios when updating a knowledge base:

SC1. Single Rule Update.
SC2. Multiple Rule Updates-Single Splitting Tree.
SC3. Multiple Rule Updates-Multiple Splitting Trees.
SC4. Simultaneous Multiple Rule Updates-Single Splitting Tree.
SC5. Simultaneous Multiple Rule Updates-Multiple Splitting Trees.

In SC1, a single rule is inserted in or removed from the source knowledge. In this case only one splitting tree (or initial merger set) is affected. In SC2, a number of rules that have the same conclusion are inserted in or removed from the source knowledge one by one. One by one means that each update (due to a rule insertion or removal) takes place at a different time. Again, only one splitting tree is affected (because of the same conclusion). In SC3, a number of rules that do not all have the same conclusion are inserted in or removed from the source knowledge one by one. In this scenario, more than one splitting tree is affected (due to the different conclusions). SC4 is the same as SC2, but all updates (insertions or removals) are simultaneously made (i.e. at the same time). The same holds for SC5 and SC3.

4.2. Rule insertion results

Tables 10 and 11 present experimental results for the update methods dealing with rule insertion for the merger sets of Table 9. In order to produce the results, some symbolic rules were removed from those merger sets and inserted afterwards one by one.

Table 10
Experimental results for rule insertions (SC1 and SC3 scenarios)

Merger set	Initial symbolic rules	Initial neurules	Merger subsets	Simple (SI) method	Total retraining (TR) method		Splitting tree (ST) method	
				Neurules	Neurules	Trained mergers	Neurules	Trained mergers
M1	7(1)	3	5	4	3	5	3	1
M2	4(1)	2	3	3	2	2	2	2
M3	9(1)	2	3	3	2	3	2	1
M4	3(1)	2	3	3	3	3	3	1
M5	2(1)	2	3	3	2	2	2	2
M6	8(1)	6	11	7	5	7	5	1
M7	19(1)	6	11	7	6	11	6	2
M8	6(1)	2	3	3	3	3	3	1
M9	3(1)	2	3	3	2	3	2	1
M10	6(1)	4	7	5	4	5	4	2
M11	3(1)	1	1	2	2	2	2	1
M12	4(1)	1	1	2	2	2	2	1
All	74(12)	33	54	45	36	48	36	16

In Tables 10 and 11, column "Initial Symbolic Rules" contains the number of the symbolic rules of the initial merger sets and (in parenthesis) the number of the symbolic rules that were inserted. For instance, entry "6(2)" at this column (in Table 11) means that the initial merger set contains six symbolic rules and two symbolic rules were inserted over those. Column "Initial Neurules" contains the number of neurules produced from the initial merger sets. Column "Merger Subsets" contains the number of merger subsets produced from the merger sets of the initial symbolic rules (including the initial merger sets themselves). "Neurules" represents the number of final neurules, produced via the three insertion methods. "Trained Mergers" represents the number of the mergers of the merger subsets that had to be trained due to the insertion of the symbolic rules.

Table 10 concerns SC1 scenario; each row, except the last one, corresponds to a SC1 case. The collective view of all rows, represented by the last row, corresponds to a SC3 scenario.

From the results in Table 10, it is obvious that SI method increases the number of neurules in NRB compared to the other two methods. The increase may not be considered as significant, if we view each insertion by itself (scenario SC1), but totally, considering all rule insertions into SRB (last row, scenario SC3), SI method produces 25% more neurules than ST method. This is graphically illustrated in Fig. 11. This increase in the number of neurules increases the inferences runtime by an average 4.1%.

On the other hand, TR method requires more training (and splitting) effort than ST method, especially when the initial merger set is split in several subsets. For instance, for M6 and M7, TR method requires training of 7 and 11 mergers respectively, whereas ST method requires training of only 1 and 2 mergers respectively. In the merger sets that are split in a few merger subsets, the difference between the two methods, in terms of the number of mergers that had to be trained, does not seem to be much enough. However, in those cases, runtime difference between the two methods is significant (see e.g. cases M3, M4, M8 and M9 in Table 12). In the cases of M11 and

Table 11
Experimental results for rule insertions (SC1, SC2 and SC3 scenarios)

Merger set	Initial symbolic rules	Initial neurules	Merger subsets	Simple (SI) method	Total retraining (TR) method		Splitting tree (ST) method	
				Neurules	Neurules	Trained mergers	Neurules	Trained mergers
M1	6(2)	3	5	5	3	10	3	3
M2	4(1)	2	3	3	2	2	2	2
M3	7(3)	2	3	5	2	10	2	5
M4	2(2)	1	1	3	2	5	2	2
M5	2(1)	2	3	3	2	2	2	1
M6	6(3)	5	9	8	5	18	5	3
M7	15(5)	4	7	9	6	53	6	13
M8	4(3)	2	3	5	3	7	3	5
M9	2(2)	1	1	3	2	5	2	2
M10	4(3)	4	7	7	4	13	4	5
M11	3(1)	1	1	2	2	2	2	1
M12	4(1)	1	1	2	2	2	2	1
All	59(27)	28	44	55	35	129	35	43

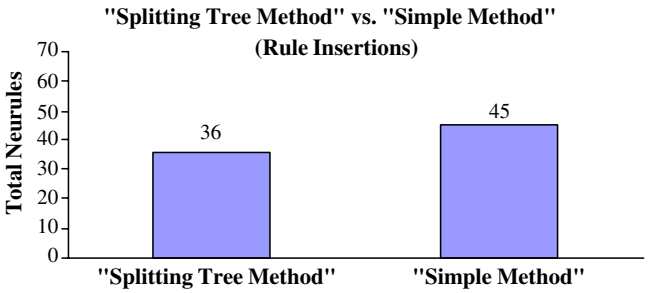

Fig. 11. Number of neurules for the cases of Table 10.

M12, ST method was used, although there was only one initial neurule. The difference between the two methods in the number of trained mergers is the result of reusing the initial merger set in the update process. Considering all rule insertions (last row, scenario SC3), TR method requires training of 200% more mergers than ST method (see Fig. 12).

Table 11 includes SC1 and SC2 cases; each row, except the last one, corresponds to either a SC1 or a SC2 scenario case. The collective view of all rows, represented by the last row, corresponds to a SC3 scenario.

From the results in Table 11, it is obvious that once again SI method increases the number of neurules in NRB compared to the other two methods. As before, the increase may not be considered as significant, if we view each initial merger set separately (scenario SC1). However, considering all rule insertions into SRB, SI method produces 57% more neurules than ST method, as illustrated in Fig. 13. Furthermore, this increase in the number of neurules increases the inferences runtime by an average 6.33%.

Table 12
Runtime results for the cases of Table 10

Merger set	ST method (ms)	TR method (ms)
M1	1.0	14.4
M2	20.5	19.9
M3	1.0	17.6
M4	20.3	478.8
M5	8.4	8.6
M6	12.5	150.0
M7	9.5	184.3
M8	10.3	102.6
M9	0.1	8.6
M10	3.7	121.8
M11	33.8	35.6
M12	30.1	32.4
All	121.4	1146.1

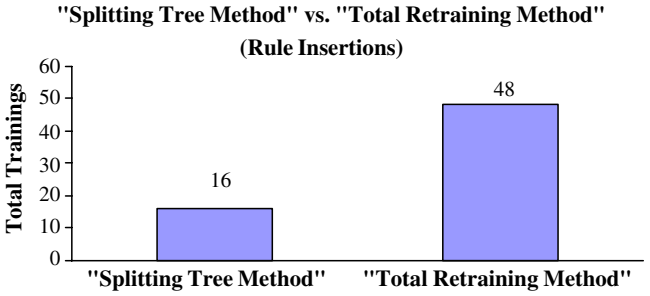

Fig. 12. Number of trainings for the cases of Table 10.

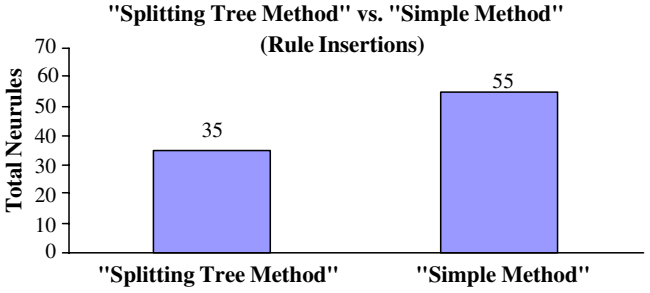

Fig. 13. Number of neurules for the cases of Table 11.

Again, TR method requires more training (and splitting) effort than ST method, especially when the initial merger set is split in several subsets. For instance, for M6 and M7, TR method requires training of 18 and 53 mergers, respectively, whereas ST method requires training of only 3 and 13 mergers, respectively. As before, in the merger sets that are split in a few merger subsets, the difference between the two methods does not seem to be significant, but the runtime difference

Table 13
Runtime results for the cases of Table 11

Merger set	ST method (SC1, SC2, SC3) (ms)	TR method (SC1, SC2, SC3) (ms)	TR method (SC4, SC5) (ms)
M1	5.0	32.0	14.4
M2	20.5	19.9	19.9
M3	1.2	51.2	17.6
M4	123.9	573.7	478.8
M5	8.4	8.6	8.6
M6	10.0	363.1	150
M7	12.0	922.6	184.3
M8	71.2	172.0	102.6
M9	4.3	14.2	8.6
M10	9.9	259.5	121.8
M11	33.8	35.6	35.6
M12	30.1	32.4	532.4
All	301.4	2456.3	1146.1

between the two methods is significant (see e.g. cases M4, M8 and M9 in Table 13 later on). Finally, considering all rule insertions into SRB (scenario SC3), TR method requires training of 198% more mergers than ST method (see Fig. 14).

Table 12 presents runtime results for ST and TR methods for the rule insertions of Table 10 (SC1 cases). It is very clear, from Table 12, that in all cases the runtime required for ST method is less (in the majority of them much less) than the runtime required for TR method. This is due, first, to the fact that ST method requires training of less mergers. Furthermore, ST method avoids training of merger sets that are higher in the splitting tree, for which training and splitting effort is more expensive than that for merger sets lower in the splitting tree (as explained in Section 4.1). As expected, runtime differences between the two methods are not proportional to their differences in the number of trained mergers.

Table 13 presents runtime results for ST and TR methods for the cases of Table 11. Table 13 includes runtime results for all scenarios. Each value of the second and third column corresponds to an SC1 or SC2 scenario case, except the last one that corresponds to an SC3 scenario case. Each value of the last column corresponds to an SC4 scenario, except the last one that corresponds to an SC5 scenario case. ST method performs much better than TR method in all cases in all scenarios.

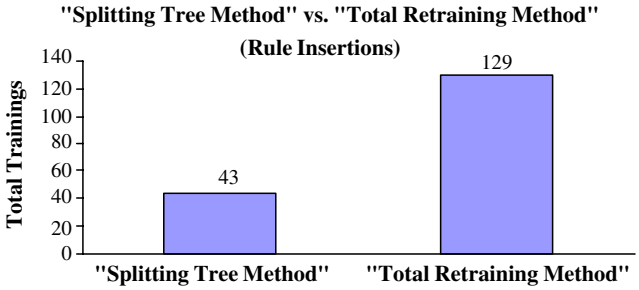

Fig. 14. Number of trainings for the insertions of Table 11.

4.3. Rule removal results

Table 14 presents experimental results related to update methods dealing with rule removal. To produce the results, some symbolic rules were removed one by one from the initial merger sets.

Column "Initial Symbolic Rules" contains the number of symbolic rules of the initial merger sets and (in parenthesis) the number of symbolic rules that were removed. For instance, entry "8(2)" of this column means that the initial merger set contains eight symbolic rules and two of those rules were removed one by one. Column "Initial Neurules" contains the number of neurules produced from the initial merger sets (including the symbolic rules to be removed). Column "Merger Subsets" contains the number of merger subsets produced from the merger sets of the initial symbolic rules (including the initial merger sets). "Neurules" represents the number of final neurules produced via the three removal methods. "Trained Mergers" represents the number of mergers that had to be (re)trained due to the removal of the symbolic rules. Each row in Table 14 corresponds to either a SC1 or a SC2 scenario case.

According to the results in Table 14, SI method increases the number of neurules in NRB by roughly 9.4% compared to the other two methods (see Fig. 15), but the difference is not as striking as in the case of rule insertions. Furthermore, TR method, as in rule insertions, requires more training (and splitting effort) than ST method (see Fig. 16), especially when the initial merger set is split in several subsets. More specifically, TR method requires 355.5% more merger trainings than ST method does.

Table 15 presents runtime results for ST and TR methods for the cases of Table 14. Table 15 includes runtime results for all scenarios. Each value of the second and third column corresponds to an SC1 or SC2 scenario case, except the last one that corresponds to an SC3 scenario case. Each value of the last column corresponds to an SC4 scenario, except the last one that corresponds to an SC5 scenario case. ST method performs much better than TR method in all cases, except two SC4 cases (M8 and M9).

Table 14
Experimental results for rule removals

Merger set	Initial symbolic rules	Initial neurules	Merger subsets	Simple (SI) method	Total retraining (TR) method		Splitting tree (ST) method	
				Neurules	Neurules	Trained mergers	Neurules	Trained mergers
M1	8(2)	3	5	3	3	10	3	1
M2	5(1)	2	3	2	2	2	2	1
M3	10(3)	2	3	3	2	10	2	6
M4	4(2)	3	5	1	1	3	1	0
M5	3(1)	2	3	2	2	2	2	0
M6	9(3)	5	9	5	5	18	5	1
M7	20(5)	6	11	6	4	53	4	9
M8	7(3)	3	5	2	2	5	2	4
M9	4(2)	2	3	1	1	3	1	1
M10	7(3)	4	7	4	4	12	4	1
M11	4(1)	2	3	3	3	2	3	1
M12	5(1)	2	3	3	3	3	3	2
All	86(27)	36	60	35	32	123	32	27

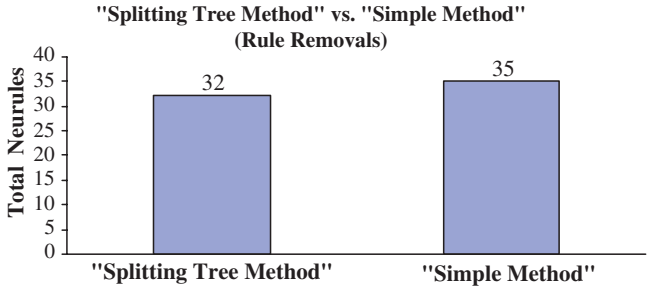

Fig. 15. Number of neurules for the cases of Table 14.

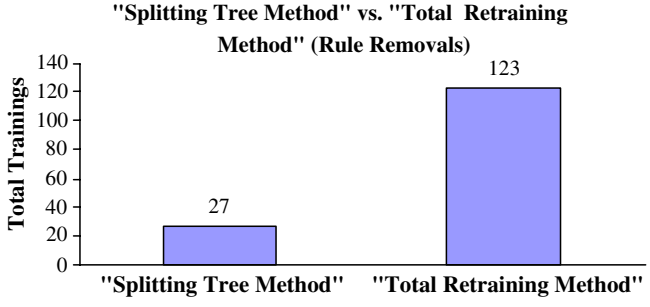

Fig. 16. Number of trainings for the cases of Table 14.

Table 15
Runtime results for the cases of Table 14

Merger set	ST method (SC1, SC2, SC3) (ms)	TR method (SC1, SC2, SC3) (ms)	TR method (SC4, SC5) (ms)
M1	0.05	34.0	15.4
M2	0.2	14	14
M3	17.4	51.8	19.6
M4	0.0	110.7	0.1
M5	0.0	1.0	1.0
M6	0.05	310.1	92.3
M7	35.3	879.5	153.3
M8	30.3	86.0	20.4
M9	4.05	4.2	0.15
M10	0.05	234.7	68.7
M11	0.0	0.4	0.4
M12	16.7	65.4	24.1
All	104.1	1791.8	409.4

5. Conclusions

In this paper, we present methods for efficient updates of a hybrid rule base. The hybrid rule base consists of neurules, a type of hybrid rules combining symbolic rules and neural networks.

The neurule base, called the target knowledge, is produced by conversion from a symbolic rule base, called its source knowledge.

The presented methods perform modifications to the target knowledge, due to insertion of a new rule in or deletion of an old rule from its source knowledge. This is done in an efficient way, that is in a way that (a) requires as little re-conversion as possible and (b) preserves the number of neurules as small as possible. This is achieved by storing information related to the conversion process in a tree, called the splitting tree. Experimental results demonstrate the benefits of using the splitting tree. The methods are also proved to be correct.

Neurules aim at improving the space and time efficiency of symbolic rules. Thus, their integration scheme gives pre-eminence to the symbolic framework. We are not aware of other similar attempts, so a comparison with other approaches is not feasible. There are other rule-based hybrid approaches that give pre-eminence to connectionism and deal with deletion and creation (or insertion) of rules within a neural network (like KBCNN [4,5] or KBANN [22,23]), but they do it for knowledge refinement purposes. A problem of those approaches is that, in most cases, almost the whole network should be re-trained, which does not happen with our approach.

Neurules can also be produced directly from empirical data/patterns [10]. In that case, there is a similar problem: to find methods for efficiently updating a neurule base, due to new empirical data/patterns. We have made an initial work on those methods [19], which could be called 'pattern-based update methods', in contrast to 'rule-based update methods', presented in this paper. Those methods are one of our future research interests in this context.

References

[1] R. Andrews, J. Diederich, A. Tickle, A survey and critique for extracting rules from trained ANN, Knowledge-Based Systems 8 (6) (1995) 373–389.
[2] B.G. Buchanan, E.H. Shortliffe, Rule-based Expert Systems, Addison-Wesley, Reading, MA, 1984.
[3] I. Cloete, J.M. Zurada (Eds.), Knowledge-based Neurocomputing, MIT Press, Cambridge, 2000.
[4] L.-M. Fu, L.-C. Fu, Mapping rule-based systems into neural architecture, Knowledge-Based Systems 3 (1990) 48–56.
[5] L.-M. Fu, A connectionist approach to rule refinement, Applied Intelligence 2 (1992) 93–103.
[6] S.I. Gallant, Neural Network Learning and Expert Systems, MIT Press, Cambridge, 1993.
[7] A.S. d'Avila Garcez, K. Broda, D.M. Gabbay, Neural-symbolic Learning Systems: Foundations and Applications, Perspectives in Neural Computing, Springer-Verlag, Berlin, 2002.
[8] A.Z. Ghalwash, A recency inference engine for connectionist knowledge bases, Applied Intelligence 9 (1998) 201–215.
[9] I. Hatzilygeroudis, J. Prentzas, Neurules: Improving the performance of symbolic rules, International Journal on AI Tools 9 (1) (2000) 113–130.
[10] I. Hatzilygeroudis, J. Prentzas, Constructing modular hybrid rule bases for expert systems, International Journal on Artificial Intelligence Tools 10 (1–2) (2001) 87–105.
[11] I. Hatzilygeroudis, J. Prentzas, Neuro-symbolic approaches for knowledge representation in expert systems, International Journal of Hybrid Intelligent Systems 1 (3–4) (2004) 111–126.
[12] M. Hilario, An overview of strategies for neurosymbolic integration, in: R. Sun, E. Alexandre (Eds.), Connectionist–symbolic Integration: from Unified to Hybrid Approaches, Lawrence Erlbaum, London, 1997.
[13] J.J. Mahoney, Combining symbolic and connectionist learning methods to refine certainty-factor rule-bases, Ph.D. Dissertation, University of Texas at Austin, 1996.

[14] K. McGarry, S. Wertmer, J. MacIntyre, Hybrid neural systems: from simple coupling to fully integrated neural networks, Neural Comput Surveys 2 (1999) 62–93.
[15] L.R. Medsker, Hybrid Neural Networks and Expert Systems, Kluwer Academic Publishers, Boston, 1994.
[16] C. Omlin, C. Giles, Rule revision with recurrent neural networks, IEEE Transactions on Knowledge and Data Engineering 8 (1) (1996) 183–188.
[17] V. Palade, D.-C. Neagu, G. Puscasu, Rule extraction from neural networks by interval propagation, in: Proceedings of the 4th International Conference on Knowledge-Based Intelligent Engineering Systems (KES 2000), Brighton, UK, August 30–September 1 2000, pp. 217–220.
[18] J. Prentzas, I. Hatzilygeroudis, Updating a hybrid rule base with changes to its symbolic source knowledge, in: Proceedings of the 15th European Conference on Artificial Intelligence (ECAI 2002), Lyon, France, July 21–26 2002, pp. 250–254.
[19] J. Prentzas, I. Hatzilygeroudis, A. Tsakalidis, Updating a Hybrid Rule Base with New Empirical Source Knowledge, in: Proceedings of the 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2002), Washington, DC, USA, November 4–6 2002, pp. 9–15.
[20] T.-S. Quah, C.-L. Tan, K.S. Raman, H.-H. Teh, B. Srinivasan, A shell environment for developing connectionist decision support systems, Expert Systems 11 (4) (1994) 225–236.
[21] R. Sun, E. Alexandre (Eds.), Connectionist–symbolic Integration: from Unified to Hybrid Approaches, Lawrence Erlbaum, London, 1997.
[22] G. Towell, Symbolic knowledge and neural networks: insertion, refinement, extraction, Ph.D. Dissertation, University of Wisconsin, Madison, 1991.
[23] G. Towell, J. Shavlik, Knowledge-based artificial neural networks, Artificial Intelligence 70 (1–2) (1994) 119–165.
[24] S. Wermter, R. Sun (Eds.), Hybrid Neural Systems, Springer-Verlag, Heidelberg, 2000.

Jim Prentzas received his Ph.D. from the Department of Computer Engineering and Informatics, University of Patras, Greece, in 2002. His Ph.D. Thesis was on ''Intelligent Hybrid Modular Systems''. He is currently member of the teaching staff at the Department of Informatics and Computer Technology, Technological Educational Institute of Lamia, Greece. He has been member of the PC of the FLAIRS conference for some years. He has participated in a number of National and European research projects. He has published over 30 papers in international journals, edited volumes, and proceedings of conferences and workshops. His main research interests are: artificial intelligence, intelligent tutoring systems, knowledge representation, web applications and geographical information systems.

Ioannis Hatzilygeroudis is currently a Lecturer at the Department of Computer Engineering and Informatics, University of Patras. He is member of the Editorial Boards of the International Journal of Artificial Intelligence Tools (IJAIT), the International Journal of Web-Based Communities (IJWBC), the International Journal of Computational Intelligence (IJCI) and the International Journal of Hybrid Intelligent Systems (IJHIS). He is the Guest Editor of a special issue of IJAIT on ''AI Techniques in Web-Based Educational Systems''. He has also been member of the PCs of several AI related conferences (e.g. IEEE, ICTAI, FLAIRS) for a number of years. He has organized a number of special/invited sessions in those conferences. He has published over 50 papers in international journals, edited volumes and proceedings of conferences and workshops. His main research interests are: artificial intelligence, hybrid knowledge representation, expert systems, knowledge engineering and intelligent educational systems.