*Ersch. in: Applications of graph transformations with industrial relevance : International Workshop, AGTIVE '99, Kerkrade, The Netherlands, September 1 - 3, 1999; proceedings / Manfred Nagl ... (ed.). - Berlin [u.a.] : Springer, 2000. - S. 263-271. - (Lecture notes in computer science ; 1779). - ISBN 3-540-67658-9*

# Learning and Rewriting in Fuzzy Rule Graphs

Ingrid Fischer[1,⋆], Manuel Koch[2,⋆⋆], and Michael R. Berthold[3,⋆⋆⋆]

[1] International Computer Science Institute, Berkeley, USA
and Chair of Programming Languages and Their Compilers
University of Erlangen-Nuremberg, Germany
idfische@informatik.uni-erlangen.de
[2] Dipartimento di SScienze dell'Informazione
Università di Roma La Sapienza, 00198 Roma
carr@dsi.uniroma1.it
[3] Berkeley Initiative in Soft Computing (BISC)
University of California at Berkeley, USA
berthold@cs.berkeley.edu

**Abstract.** Different learning algorithms based on learning from examples are described based on a set of graph rewrite rules. Starting from either a very general or a very special rule set which is modeled as graph, two to three basic rewrite rules are applied until a rule graph explaining all examples is reached. The rewrite rules can also be used to model the corresponding hypothesis space as they describe partial relations between different rule set graphs. The possible paths, algorithms can take through the hypothesis space can be described as application sequences. This schema is applied to general learning algorithms as well as to fuzzy rule learning algorithms.

## 1 Introduction

Building models from data has started to raise increasing attention, especially in areas where a large amount of data is gathered automatically and manual analysis is not feasible anymore. Also applications where data is recorded online without a possibility for continuous analysis are demanding for automatic approaches. Examples include such diverse applications as the automatic monitoring of patients in medicine (which requires an understanding of the underlying behavior), optimization of industrial processes, and also the extraction of expert knowledge from observations of their behavior. Techniques from diverse disciplines have been developed or rediscovered recently, resulting in an increasing set of tools to automatically analyze data sets. Most of these tools, however, require the user to have detailed knowledge about the tools' underlying algorithms, to fully make use of their potential. In order to offer the user the possibility to

⋆ I. Fischer was supported by a postdoc "Gemeinsamen Hochschulsonderprograms III von Bund und Ländern" stipend from the DAAD.
⋆⋆ M. Koch was supported by the TMR network GETGRATS
⋆⋆⋆ M. Berthold was supported by DFG-Grant Be1740/7–1.

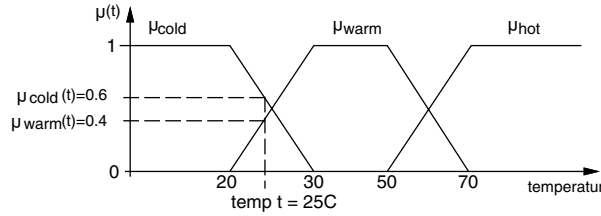


**Fig. 1.** A linguistic variable *temperature* with three linguistic values (described through fuzzy sets) *cold*, *warm*, and *hot* and the degrees of memberships for a certain temperature $t$.

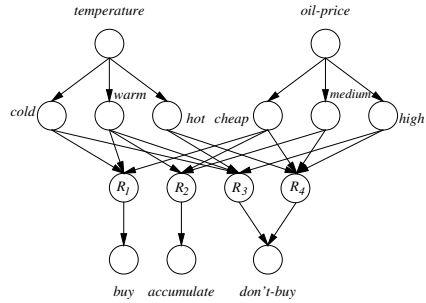


**Fig. 2.** Rule graph for the rules given in example 1.

explore the data, unrestricted by a specific tool's limitations, it is necessary to provide easy to use, quick ways to give the user first insights. In addition the extracted knowledge has to be presented to the user in an understandable manner, enabling interaction and refinement of the focus of analysis.

Learning rules from examples is an often used approach to achieve this goal. Over the years different rule learning algorithms have been developed as e.g. [7,9,6,5]. For fuzzy rules, training algorithms are described in [1,10,15,3]. Overviews for both fields are given in [2].

## 2 Fuzzy Rules

*Example 1.* In this paper we will use an example based on fuzzy rules [16,17] to explain the basic concepts. First the data used will be explained as well as possible rules describing this data. Our examples handles the question when oil should be bought depending on the current weather and the current oil-price. Three linguistic variables are used to describe these conditions. The linguistic variable *temperature* with its linguistic values *cold*, *warm* and *hot* is shown in Fig. 1. Similarly the linguistic variable *oil-price* with its values *cheap*, *medium* and *high* and *buy-ranking* with *buy, accumulate, don't buy* are used. Rules making use of these variables are the following:

$\mathcal{R}_1$: `if` *temperature* `is` (*cold* `or` *warm*) `and` *oil-price* `is` *cheap*
`then` *buy-ranking* `is` *buy*
$\mathcal{R}_2$: `if` *temperature* `is` *warm* `and` *oil-price* `is` (*cheap* `or` *medium*)
`then` *buy-ranking* `is` *accumulate*
$\mathcal{R}_3$: `if` *oil-price* `is` *high* `then` box*buy-ranking* `is` *don't buy*
$\mathcal{R}_4$: `if` *temperature* `is` *hot* `then` *buy-ranking* `is` *don't buy*

Simple IF–THEN–rules together with conjunctions and disjunctions are used. These rules can be transformed to a rule graph as shown in Fig. 2. Each linguistic variable used as input and their corresponding linguistic values, each linguistic values describing the result as well as each rule is modeled as a node. In our running example there are nodes for *temperature*, *cold*, *warm*, *hot*, *oil-price*, *cheap*, *medium*, *high*, *don't-buy*, *accumulate*, *buy* and nodes modeling the four rules $\mathcal{R}_1$, $\mathcal{R}_2$, $\mathcal{R}_3$, $\mathcal{R}_4$. Edges connect rule nodes to their input and output nodes as well as linguistic variables to their possible values. Other graph based descriptions of rules are e.g. (Fuzzy) Petri Nets [4,13] having also the advantage that the execution of rules can be modeled with rewrite rules. But for the sake of clarity (and space) a "smaller" graphical representation was chosen here.

In the following, different training algorithms generating rules from examples are described. As description language graphs as shown in Fig. 2 are used together with simple graph rewrite rules. The special rewrite formalism used is not further described since it is possible without problems to use different formalisms as described in [14]. The next sections will deal with training algorithms starting bottom-up from a special graph (i.e. the most specific rules) and top-down from a very general graph (the most general rules). Based on rewrite rules the set of all possible rule graphs, the hypothesis space, can be described.

## 3 Bottom Up Training

Starting with a very special graph and generalizing it until it covers all given training examples is the basic idea for bottom up training. The most special rules for a given set of examples are the examples itselves. It is easy to transform an example into a rule and this rule then covers nothing else than its underlying example. In [15] such an algorithm was introduced.

*Example 2.* For our running example we take the data given in Fig. 3 (left). Four possible data points are shown. For the sake of simplicity no actual numbers are given here. It is only indicated in what possible linguistic value this number falls. Transferring each of the above examples into a special rule leads to a graph as shown on the right hand side of Fig. 3. E.g. the data example *cold - cheap - buy* leads to the following rule:

$\mathcal{R}$: `if` *temperature* `is` *cold* `and` *oil-price* `is` *cheap*
`then` *buy-ranking* `is` *buy*

In Fig. 3 this rule is marked *R*.

| Temperature | oil-price | buy-ranking |
|---|---|---|
| cold | cheap | buy |
| warm | medium | accumulate |
| warm | high | don't buy |
| hot | medium | don't buy |

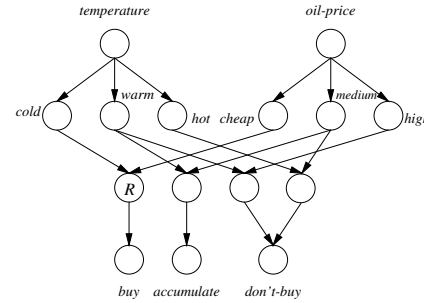


**Fig. 3.** Four data points for training (left) and the corresponding rule graph (right)

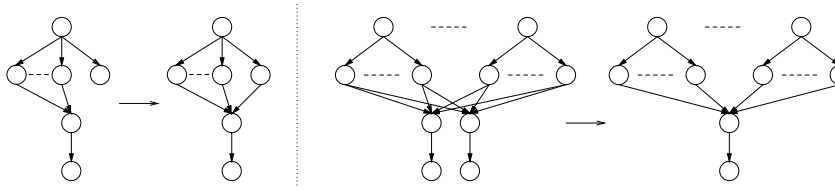

**Fig. 4.** Generalizing (left) and merging (right) in rule graphs.

As these kind of example-based rule graphs do not generalize at all, it is useful to change their structure to respond also to other inputs while still classifying the examples correctly. Two kinds of operations can be performed to change the rule graph. First the input to a rule can be generalized. In this case a new edge is inserted pointing from the possible new input to a rule i.e. a node representing a linguistic value e.g. *hot*, to the node representing the rule itself. This rewriting rule is shown in Fig. 4 (left side). A negative application condition ensuring that there is at most one edge between nodes is omitted.

*Example 3.* To the graph in Fig. 3 this rule could be applied ten times leading to a graph where each of the four rules has all possible inputs. As there are only three output values there are two equal rules handling the output *don't-buy*. These rules can be merged with the help of the rule shown in Fig. 4 on the right. Merging two rules means that two identical rules are merged into one rule or that one of the identical rules is deleted. This leads to the graph as shown in Fig. 5. This graph shown in Fig. 5 is the most general possible graph which gives a positive output for all three output classes no matter what the input is. The graph given in Fig. 2 handles the given examples also correctly. It can be reached by applying the generalization rule once to the first and second rule node and twice to the third and fourth rule node starting from the graph given in Fig. 3, right. The merge rule is not applied at all.

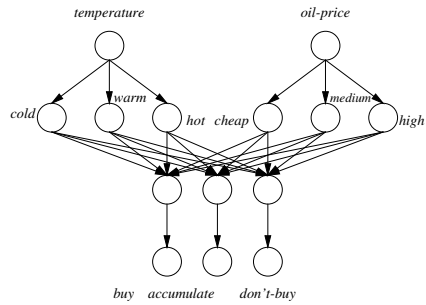


**Fig. 5.** A Rule Graph containing one Rule for each Output Class

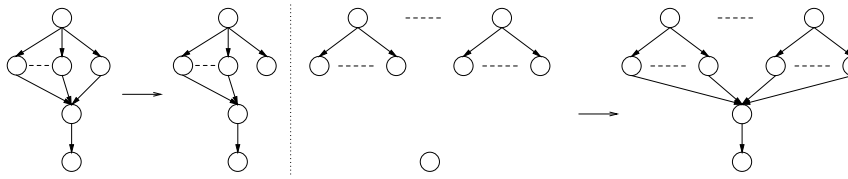

**Fig. 6.** Shrinking (left) and Committing (right) in Rule Graphs

## 4 Top Down Training

Fig. 5 serves as starting point for algorithms like [3]. In contrast to the last section the graph must now be specialized since most general rules are used which do not handle the examples correctly. In [3] two operations are suggested: A *shrink* operation minimizing the input into a rule i.e. deleting an edge pointing from a linguistic value to a rule node. Additionally a *commit* inserting a very general new rule, a rule that has all possible inputs. Both rules are given in Fig. 6. Taking Fig. 5 and the examples given in Table 3, one *commit* and ten *shrink* leads to the rule graph in Fig. 2 classifying the examples correctly. Of course this is just one possible path leading to one possible rule graph. Other paths and other rule graphs are possible. One of the possible final results of applying *shrink* and *commit* to the start graph as shown in Fig. 5 is again shown in Fig. 2.

## 5 Organizing Models

With the graph shown in Fig. 5 being the most general rule graph having *true* as its only possible output, it is obvious that Fig. 3 does not contain the most specific graph. This most specific graph is shown in Fig. 7 containing no rule at all. Following [12] the different rule graphs can be organized in the so called

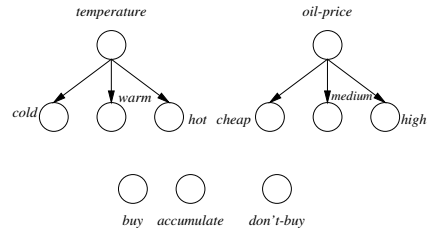


**Fig. 7.** The bottom element of the rule graphs.

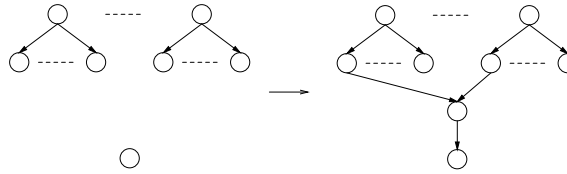

**Fig. 8.** A rule is inserted taking exactly one input from each variable.

*hypothesis space* between the top and the bottom element. In between these two, all possible rule graphs can be organized as follows:

Starting from the bottom element, the empty graph, three operations are necessary to build up the complete set of rule graphs:

- An *insert-special* rule as shown in Fig. 8 inserting the most special rule into the rule graph having one input from each variable as e.g. *temperature* or *oil-price*.
- A *generalize* rule as shown in Fig. 9 doing nothing else than extending the input of a rule by inserting an edge from a variable like e.g. *hot*, *medium*, *cold* to a node representing a rule. Of course it must be ensured that such an edge does not exist already.
- A *delete-general* rule as shown in Fig. 10, which is a rule that deletes the most general rule, a rule that has input from every possible value. Nevertheless it must be ensured that there is another rule covering the same output class otherwise it would be possible to get back to the bottom element in a cycle.

Going the other direction from the top to the bottom element, these three rules can be used inversely as *specialize*[1], *delete–special* and *insert–general*. Taking for example the algorithm presented in Section 4 it only uses the *specialize* and *insert-general* rule named there *commit* and *shrink*. In other cases several of these small steps must be executed sequentially without interruption to form one single step the algorithm is using. For example in Section 3 a rule is inserted

[1] This rule is often called *dropping* rule in the literature [11].

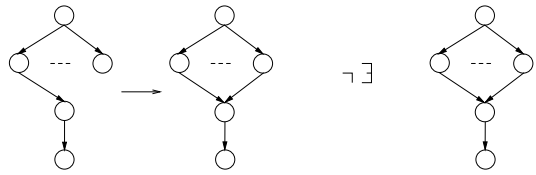

**Fig. 9.** The input to a rule is generalized by inserting a new edge between a node modeling a variable and a node modeling a rule.

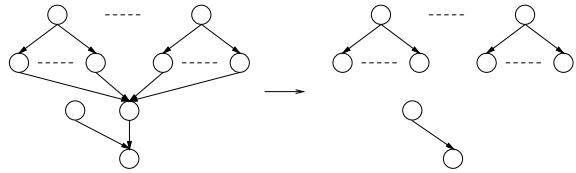

**Fig. 10.** A most general rule having input from each possible value is deleted. It must be ensured that there is one remaining rule for the output class.

for each example. This bigger step consists of one *insert-special* and several *generalize* rule applications.

Nevertheless infinitely many rule graphs can be created by subsequently inserting new rules. Semantically this means that equal rules exist in the graph. The number of really different rule graphs can be calculated as follows. Having $n$ input classes with $m_i, 1 \leq i \leq n$ possible values then there are

$$R = \prod_{i=1}^{n} \sum_{j=1}^{m_i} \binom{m_j}{j}$$

different rules and $2^R$ different rule graphs for only one output class. It is with the specific algorithm used that one has to make sure that it actually terminates.

## 6 What Can Be Done?

With the hypothesis space defined as described the following issues can be explored further:

- Having a theoretically founded description method, it is straightforward to use it for further theory based modeling and proofing. For example termination of training algorithms can be shown based on ideas developed for general rewriting. By mapping graphs onto elements of a terminating partial order and by showing that the application of rewrite rules to graphs only makes them smaller along the lines of this partial order termination can be

shown. E.g. the termination of the algorithm given in [3] can be proven along the same line as shown for a similar Neural Network algorithm in [8].
- The version space [12] contains all rule graphs that describe the training examples correctly. Assume that two graphs are in relation if they handle the same set of training examples correctly. This equivalence relation can be used to describe confluence. If an algorithm can be shown to be confluent with respect to this equivalence relation it always leads to a result modeling the training data correctly. This does not mean that always the same result is reached. The goal is just to show that one possible rule graph can be constructed ensured by the equivalence relation. Results obtained in graph rewriting help to proof (local) confluence. When taking the Double-Pushout Approach into account ([14], Chapter 1) theorems dealing with parallel independence are useful.
- Data can be noisy so that it is useful not to take certain examples into account when building the rule model. In that case the hypothesis space and the version space do not differ much. This leads to hierarchical spaces. It is an interesting question how these spaces are related.
- A practical application tool support must be provided. In a rapid prototyping system for rule generation few basic transformation rules must be provided together with e.g. application conditions and control flow specifications. With the help of a visualization for a smaller set of testing examples it can be visualized how fast a given algorithm converges to a possible rule graph.

Whereas the first two points represent actual developments, point three and four are future work.

## References


1. S. Abe and M. S. Lan. A method for fuzzy rules extraction directly from numerical data and its application to pattern classification. *IEEE Trans. Fuzzy Systems*, 3(1):18–28, 1995.
2. M. Berthold and D. J. Hand, editors. *Intelligent Data Analysis: An Introduction.* Springer Verlag, 1999. 264
3. M. R. Berthold and K. P. Huber. Constructing fuzzy graphs from examples. *Intelligent Data Analysis*, 3(1):37–54, 1999. 264, 267, 270
4. J. Cardoso and H. Camargo. *Fuzziness in Petri Nets.* Springer–Verlag, 1999. 265
5. P. Clark and R. Boswell. Rule induction with CN2: Some recent improvements. In Y. Kodratoff, editor, *Proceedings of the European Working Session on Learning: Machine Learning (EWSL-91)*, volume 482 of *LNAI*, pages 151–163. Springer Verlag, 1991. 264
6. W. W. Cohen. Fast effective rule induction. In *Proc. 12th International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995. 264
7. P. Domingos. Efficient specific-to-general rule induction. In E. Simoudis, J. W. Han, and U. Fayyad, editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, page 319. AAAI Press, 1996. 264

8. I. Fischer. *Describing Neural Networks with Graph Transformations*. PhD thesis, University of Erlangen–Nuremberg, 1999. 270
9. R. Holve. Investigation of automatic rule generation for hierarchical fuzzy systems. In *Proceedings of FUZZ IEEE at the 1998 IEEE World Congress on Computational Intelligence*, pages 973–978, 1998. 264
10. H. Ishibuchi, K. Nozaki, N. Yamamoto, and H. Tanaka. Selecting fuzzy if-then rules for classification problems using genetic algorithms. *IEEE Transactions on Fuzzy Systems*, 3(3):260–270, 1995. 264
11. R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors. *Machine Learning: An Artificial Intelligence Approach*, volume 1. Morgan Kaufmann, San Mateo, California, 1983. 268
12. Tom Mitchell. *Machine Learning*. McGraw-Hill Series in Computer Science, 1997. 267, 270
13. W. Reisig. *Petri Nets (an Introduction)*. Number 4 in EATCS Monographs on Theoretical Computer Science. Springer, Berlin, 1985. 265
14. Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation. Vol. I: Foundations*. World Scientific, 1997. 265, 270
15. L. X. Wang and J. M. Mendel. Generating fuzzy rules by learning from examples. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(6):1414–1427, 1992. 264, 265
16. L. A. Zadeh. Soft computing and fuzzy logic. *IEEE Software*, pages 48–56, 1994. 264
17. L. A. Zadeh. Fuzzy logic and the calculi of fuzzy rules and fuzzy graphs: A precis. *Multiple Valued Logic*, 1:1–38, 1996. 264