Modeling constraints with the fuzzy logic inference processor FLIP++

Markus Bonner Stefan Mayer Andreas Raggl
Wolfgang Slany*
E184/2, TU Wien, A-1040 Vienna, Austria, Europe
E-Mail: {bonner|mayer|raggl|wsi}@dbai.tuwien.ac.at
http://www.dbai.tuwien.ac.at/pub/slany.html#sflip

February 8, 1996
Topic: Decision Support Systems

FLIP++ is a fuzzy logic inference processor library. This C++ library was developed to do calculations with uncertain data and priorities. It evaluates various types of rules of different importance from a hierarchy of rules for uncertain or precise data. It evaluates only those parts of the hierarchy needed for the result. The result itself has two forms: first as a list of membership functions, and second as a crisp value usable for fuzzy control. The library also contains the whole functionality needed to manage one or more sets of rules on all data. It is built to be easily adapted to new applications. Due to its quite general concept of classes, it is possible to include it for the purpose of fuzzy computations in other systems.

1 Qualitative knowledge representation

It is often not possible to express knowledge in numbers or other forms usable on a computer. If the knowledge consists only of terms in human language, like 'large' or 'small', it is impossible to do calculations with these terms. With fuzzy logic such representation of knowledge is possible. A second advantage is that, like in the human usage of such terms, the exact meaning and the range of values is not the same but depends on the context in which the terms are used. The connection to a context can be done later, or the context can be adapted to new circumstances. Therefore, such qualitative variables have no numbers as possible values, but just terms, as mentioned above, and are an ideal representation for vague or uncertain knowledge.

In this paper the representation of knowledge will be shown in an example that is typical for the use of the whole *FLIP++ project, which FLIP++ is a part of. The example consists of constraints needed for scheduling a steelmaking plant. Here, three different types of constraints are used: first the chemical compatibility constraints, which will be shown in detail, second the temporal constraints for the representation of delivery dates that have to be fulfilled, and third the capacity constraints for human, machinery and other common restrictions. Each of these three constraints consists of many different subconstraints. The chemical compatibility constraint, for example, consists of constraints for every measured chemical element. It is derived from the difference between the contents of an element in the old and the new load. This difference in content is also a qualitative variable with terms as value to express the difference. Here we will use the percentage difference for the chemical element Nickel (Ni) as qualitative variable with the allowed terms: *negative*, *negative_small*, *zero*, *positive_small* and *positive*. Figure 1 shows the corresponding membership functions for such a representation. We will later derive from this variable the constraint for the compatibility for Ni by using rules.

2 Quantitative knowledge representation

For the evaluation of any rules using qualitative variables, we give the used terms some quantitative representation. This means to put them in some form with which ordinary calculation can be done. Each term is described by some membership function, which states the membership of this term for each crisp value in some range of possible values. The membership function can have values from 0 to 1. A value of the membership

*Please send all correspondence regarding this paper to Wolfgang Slany.

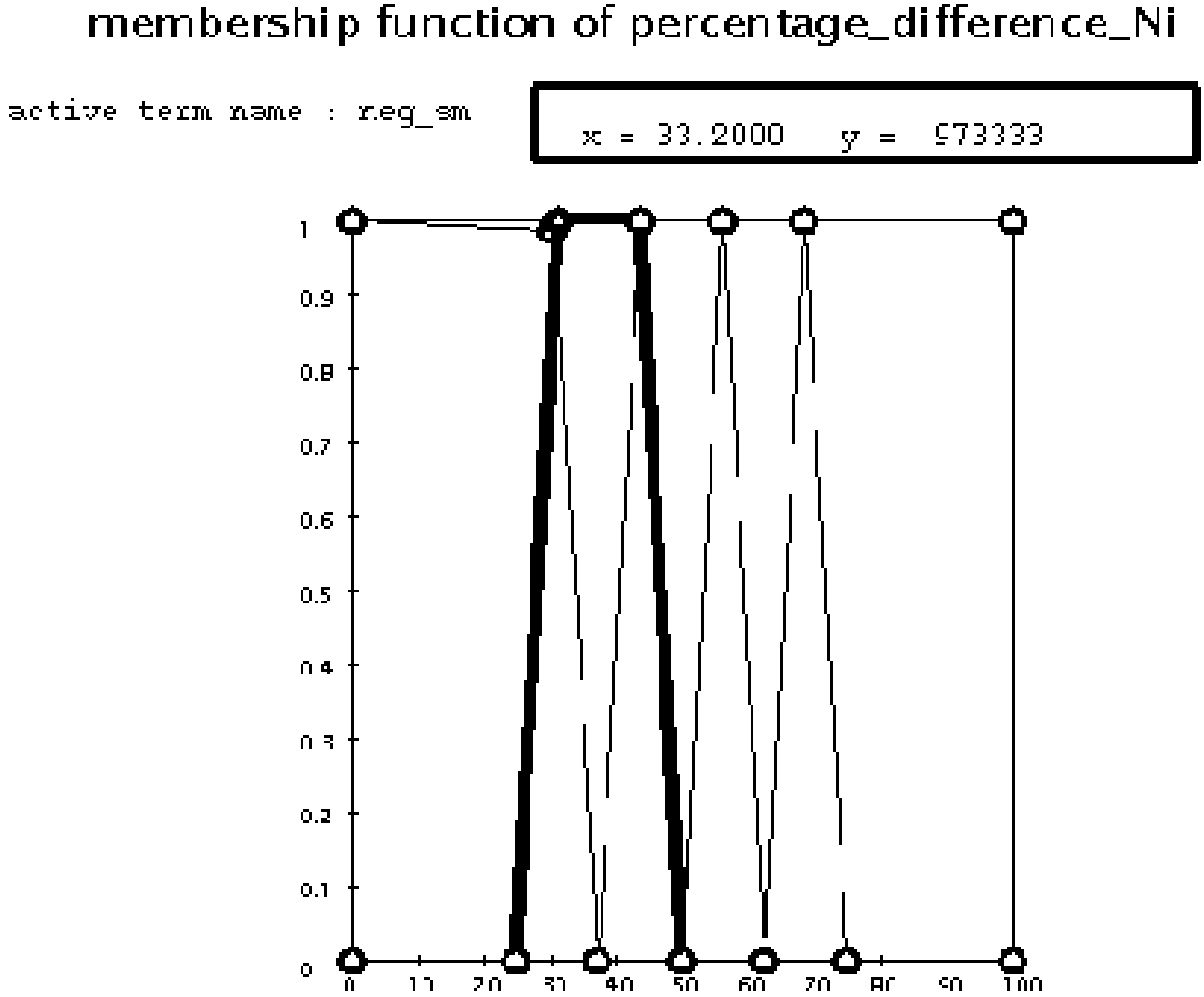

Figure 1: User Interface Percentage Difference Nickel

function of 1 means that the concerned crisp values fully represent the term. A membership function value of 0 states that the crisp values in these areas have nothing to do with this term. All other function values between 1 and 0 express a partial membership of the crisp value to the term.

3 Instantiation of variables

When any rules should be used to evaluate some given knowledge, it is necessary to put this knowledge into the used variables. The variables on the left side of any rule have to be instantiated in any form to get a result out of the rule.

One way is to directly give the variable a value by setting a certain crisp value or by attaching an instantiated membership function to it. The other way is to write some rules that create an instantiation for this variable.

When a variable is evaluated, the system will first check whether the variable is already instantiated. If not, it will try to find any rule that evaluates this variable.

4 Relations within qualitative knowledge

The representation of knowledge in form of qualitative variables alone will not deliver results. To gain new knowledge from what we know, it is necessary to include the links between qualitative variables by formulating rules. By applying these rules, new results are derived from instantiated variables and written into the dependent variable, which is to be evaluated. The rules contain the known relations concerning two or more variables. All

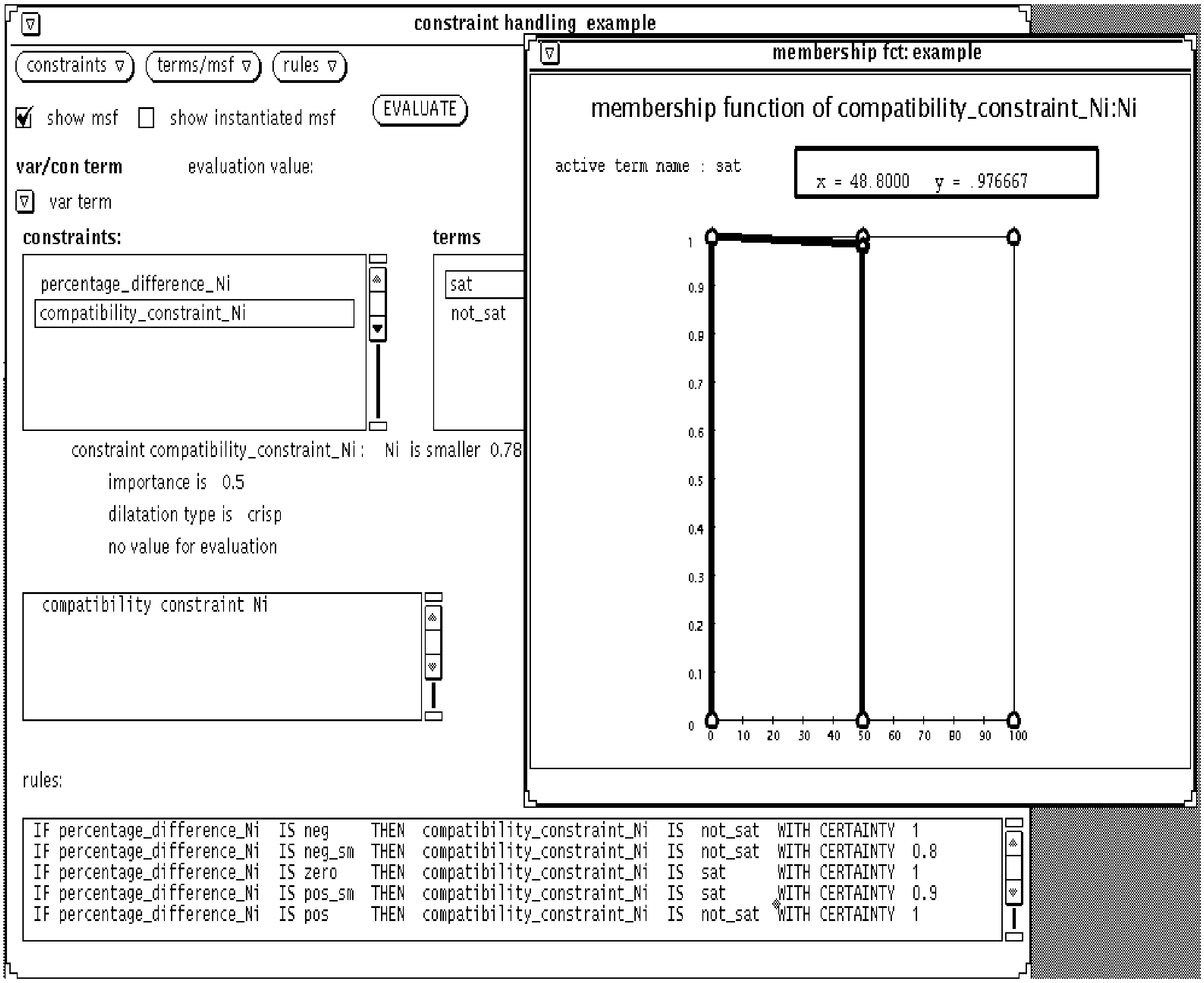

Figure 2: User Interface with Rules and Variables

clauses on the left side of such a rule consist of a variable and a term. One clause is evaluated by the use of the quantitative representation of its term. All the results of the left side clauses are aggregated, and, in a further step, used to generate some new knowledge in the variable on the right side.

4.1 The relation in the example

In addition to the qualitative variable used above, a second variable representing a constraint for the satisfaction of the content is used. The compatibility constraint for nickel (Ni) has the set of terms consisting of *satisfied* and *not_satisfied* as possible values. The term *satisfied* means that the content of nickel fulfills the constraints expressed in the rules, whereas the term *not_satisfied* stands for the opposite. Figure 2 shows the corresponding rules and membership functions when defining these terms in FLIP++.

4.2 Constraint tree - modularity

It is easier and safer to create a complex system of constraints, or other rules, in different modules and then to combine them. A main intention of fuzzy logic is to bring computer systems nearer to reality and human thinking. This makes more sense than trying to force reality into a computer-readable form. The latter approach is almost always complicated and of uncertain success. It also has less to do with human problem solving techniques.

Building smaller modules is easier to handle, and small modules can be verified with less effort. For more complex systems it is recommended to build a hierarchy of modules. This makes even complicated problems

Figure 3: General tree of constraints

understandable for users.

In FLIP++ each level in a hierarchy has the same methods and is directly accessible and evaluable. The hierarchy is build up automatically through the implicit dependencies in the rules. It can be created explicitly by the use of smaller modules which can be combined to a more complex module.

The steelplant example consists of many small modules or set of rules for the single constraints. All these are combined into several modules: the first one for the chemical compatibility constraints, the second one for the temporal delivery date constraints, and the last one for the capacity constraints. Their join builds the overall set of rules or the highest level in the hierarchy, as exemplified in Figure 3.

To build up such a complex set of constraints requires the ability to combine small set of rules and variables, i.e., small modules, to a greater set of rules. This was one of the major aims of FLIP++.

A second requirement is also derived from the use of hierarchies. Reasoning with such variables and rules needs associativity in the evaluation of rules. Otherwise, the order of single modules in one level of the hierarchy would influence the result.

5 Operators

Fuzzy operators are used for the aggregation of rules combining more than one variable to a variable of a higher level in the hierarchy. The above mentioned need for associativity is one of two major requirements for operators.

The second one is that it should represent an aggregation that is similar to the one which a human uses. Since the rules represent the human sight of a problem, the combination of such knowledge should be done in a way comparable to human thinking. Human understanding of operators like 'and' or 'or' is not the strict logical one, but lies between these to extremes. Such aggregations can be done with compensatory operators. The drawback of such compensatory operators is that they are not associative [3], and, therefore, would not stand our requirements.

FLIP++ allows to use a plethora of different aggregation operators. Notably, FLIP++ allows the use of averaging aggregation operators that are associative, such as the one described by Klement et al. [2]. The concrete operator used in FLIP++ is described by the authors in [1].

6 Combination of already represented knowledge

A major requirement for modularity is to be able to handle single parts of the used hierarchy. Thus it is possible to build up a complex system of qualitative variables and rules in small and verifiable steps that are easy to understand.

Two or more single sets of rules with the concerned variables can be combined by adding the variables and rules from one such system to the other. Alternatively, all can be put into a new module. When they are put together in one system, there is still no connection between the used variables and rules. The linking of the seperated parts is done by adding new rules that express their relationships. This is the same step as just combining qualitative variables. In this way complex systems of variables and rules can be build. The only restrictions for the size of the module come from the computer system on which FLIP++ is running.

To compensate for these restrictions, it is possible to evaluate the single modules. The connections between them are then expressed in a new module with new rules and the output variables from the other modules. In this case, it is important to evaluate the overall module again after changes and new evaluations in the lower modules have been done.

7 Reasoning by evaluating the knowledge

The reasoning in FLIP++ is done by evaluating the rules. The rules represent the knowledge about the connections of the qualitative variables, which in turn represent themselves the static knowledge of the domain in form of terms and their quantitative representation. The evaluation is done for one single variable. All rules that contribute reasons why this one variable should get a certain fuzzy value are searched among all rules in the set of rules. For all variables the same is done recursively. So all the static knowledge and all the known relations for the resulting variable are searched in the hierarchy and then evaluated. During the evaluation, all the uncertainties about variables and the uncertainties about the relations are used in the calculations. The rules can be evaluated by applying fuzzy methods on the quantitative representations of the variables. As a result, a degree of membership is attached to each possible term of the resulting variable. For control applications, this representation of the result can be interpreted in different ways to get some crisp value for controlling any machinery.

8 Adaptations and further developments

To make the FLIP++ library fit better for different applications, there is the possibility to add new operators. To combine it with other systems one can easily put a new interface as front end since FLIP++ is written in an object oriented language.

The next step of the project will be to enhance FLIP++ with a neural net to find better solutions for a given problem.

References

[1] Markus Bonner, Stefan Mayer, Andreas Raggl, and Wolfgang Slany. FLIP++: A fuzzy logic inference processor library. In *Workshop Notes of the IJCAI'95 Workshop on Fuzzy Logic in Artificial Intelligence*, pages 83–92, Montreal, Canada, August 1995. URL: ftp://ftp.dbai.tuwien.ac.at/pub/papers/slany/ijcai95wsflai1.ps.gz.

[2] Erich Peter Klement, Radko Mesiar, and Endre Pap. Associative compensatory operators. In Mario Fedrizzi, Erich P. Klement, Aldo Ventre, and Alessandro Zorat, editors, *Proceedings of CIFT'94: Current Issues in Fuzzy Technologies: Decision Models and Systems*, Trento, Italy, June 1994.

[3] Herbert Toth. Epistemische und formale Grundlagen der Fuzzy Set Theorie. Lecture notes of the course given at the Technical University of Vienna, 1993. In German language.