FLIP++
A fuzzy logic inference processor library

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

January 28, 1995

Abstract

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 useable 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 the classes it is possible to include it for the purpose of fuzzy computations in other systems.

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

1 FLIP++ as part of a scheduling project

FLIP++ is useable as a standalone fuzzy control module together with the user interface from the InterFLIP++ library. Alternatively it can be linked together with any other software where fuzzy calculations are needed.

Moreover, FLIP++ is part of the *FLIP++ library [2] for real-world decision making. This allows optimizing under vague constraints of different importance using uncertain data, where compromises between antagonistic criteria can be modeled. Typical application areas include scheduling, design, configuration, planning, and classification. *FLIP++ is composed of the following layered sub-libraries:

- FLIP++: the basic fuzzy logic inference processor library.
- ConFLIP++: the static fuzzy constraint library.
- DynaFLIP++: the dynamic fuzzy constraint generation library.
- DomFLIP++: the domain knowledge representation library.
- OptiFLIP++: the heuristic optimizing library; several heuristics have so far been implemented and tested, namely:
  - a tabu list min-conflicts repair based hill climbing heuristic,
  - a min-conflicts repair based iterative deepening heuristic,
  - a min-conflicts repair based random search hill climbing heuristic, and
  - a min-conflicts repair based genetic algorithm heuristic.
- CheckFLIP++: the knowledge-change consistency checker library that also allows to fine-tune the configuration parameters of a problem.
- InterFLIP++: the userinterface for all other libraries.
- DocuFLIP++: the on-line documentation available separately for end-users, knowledge-engineers, and programmers, and accessible via the World-Wide-Web as HTML documents.

2 Representation of knowledge

2.1 Qualitative representation

It is often not possible to express knowledge in numbers or other computerusable form. If the knowledge consists only of terms in human language, like 'large' or 'small', it is impossible to do calculations with these terms. It is the major advantage of fuzzy logic that this representation of knowledge is usable. A second one is that, like in the human usage of such terms, the exact meaning and the range of values is not common but depends on the context in which the terms are used. And the connection to a context can be done later or it 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, from 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 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.

2.2 Quantitative 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 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.

2.3 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 attach an instantiated membership function to it. The other way is to

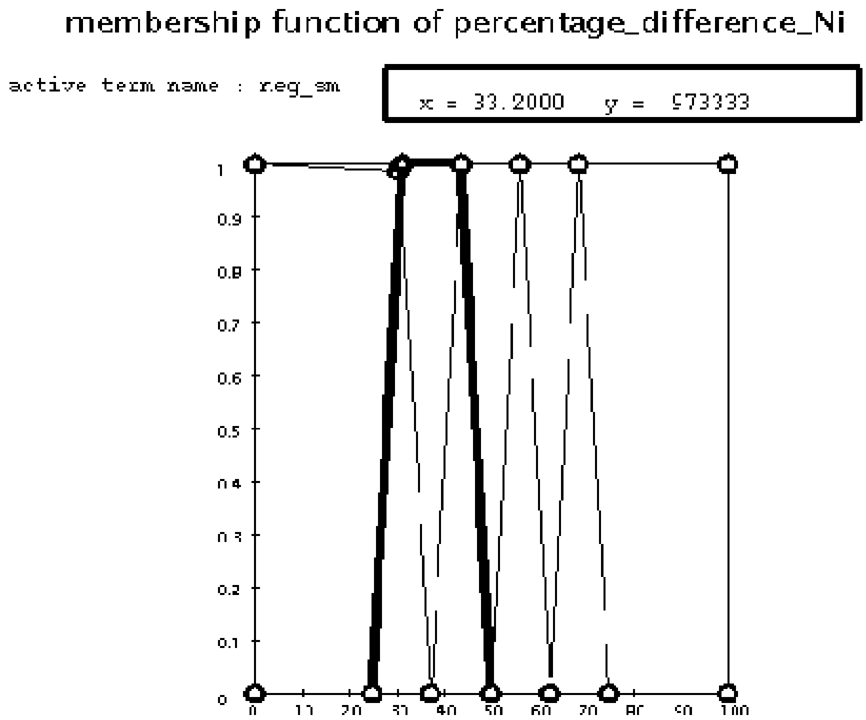

Figure 1: User Interface Percentage Difference Nickel

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.

3 Relations within the qualitative knowledge

Only the representation of knowledge in form of qualitative variables will deliver no 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 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

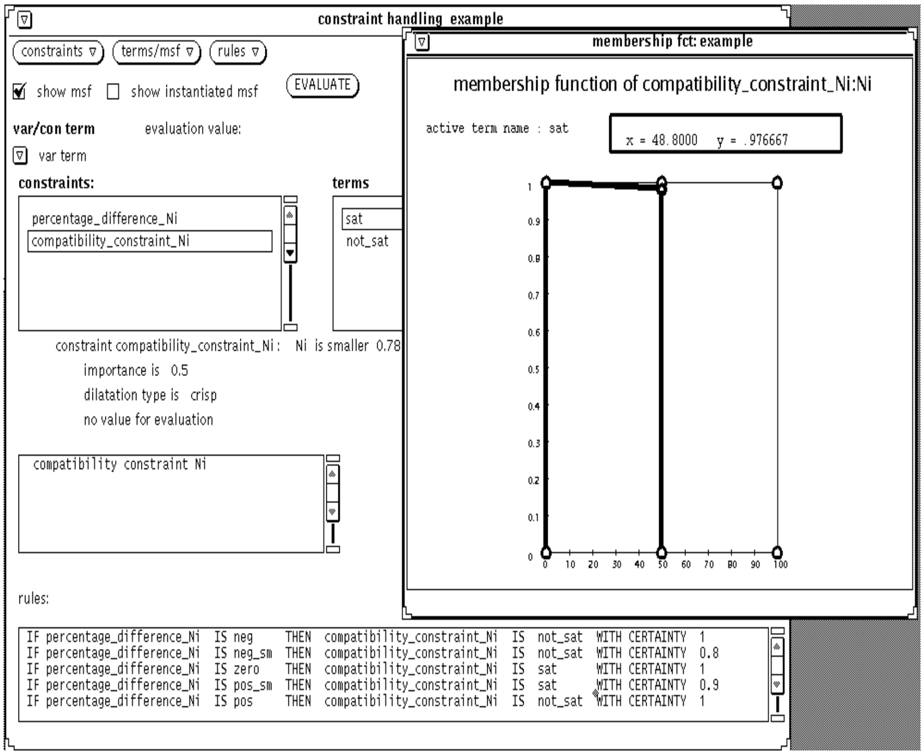

Figure 2: User Interface with Rules and Variables

the variable on the right side.

3.1 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++.

3.2 Constraint tree - modularity

It is easier and safer to create a complex system of constraints, or other rules, in different modules and then combine them. A main intention of fuzzy logic is to bring computer systems nearer to reality and human thinking. This makes

Figure 3: General tree of constraints

more sense than trying to force reality in a computer-readable form. The latter approach is almost always complicated and of uncertain success. And it 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 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. Explicitly it can be created 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 from the system 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.

4 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.

4.1 ACOtan - an Associative Compensatory Operator

A special operator that holds both above explained requirements is an associative compensatory operator designed according to a paper by Erich Peter Klement, Radko Mesiar, and Endre Pap [1]. From their Theorem 1 they derive their Definition 1 for an associative compensatory operator with multiplicative generator f.

Theorem 1 *Let* $C : [0,1]^2 \to [0,1]$ *be a binary operation on* $[0,1]$.
The following are equivalent:

(a) The operation C *is continuous on* $[0,1]^2 \backslash \{(0,1),(1,0)\}$ *, commutative, associative, non-decreasing in both components, and satisfies the cancellation law on* $]0,1[$ *and the boundary conditions* $C(0,0) = 0$ *and* $C(1,1) = 1$.

(b) There exists a closed subinterval I *of* $[0,+\infty]$ *and a continuous, strictly increasing and surjective function* $f : [0,1] \to I$ *with* $f(\{0,1\}) \epsilon \{0,1,+\infty\}$ *such that for all* (x,y) *element* $[0,1]^2 \backslash \{(0,1),(1,0)\}$

$$C(x,y) = f^{-1}(f(x) \cdot f(y)).$$

Definition 1 *Let* $f : [0,1] \to [0,+\infty]$ *be a continuous, strictly increasing function with* $f(0) = 0$ *and* $f(1) = +\infty$. *Then the binary operation* $C : [0,1]^2 \to [0,1]$ *defined by*

$$C(x,y) = \begin{cases} f^{-1}(f(x) \cdot f(y)) & if\{x,y\} \neq \{0,1\} \\ 0 & otherwise, \end{cases}$$

is called an associative compensatory operator with multiplicative generator f.

The duals of such operators are shown to be again associative compensatory operators, and a characterization of self-dual operators is given.

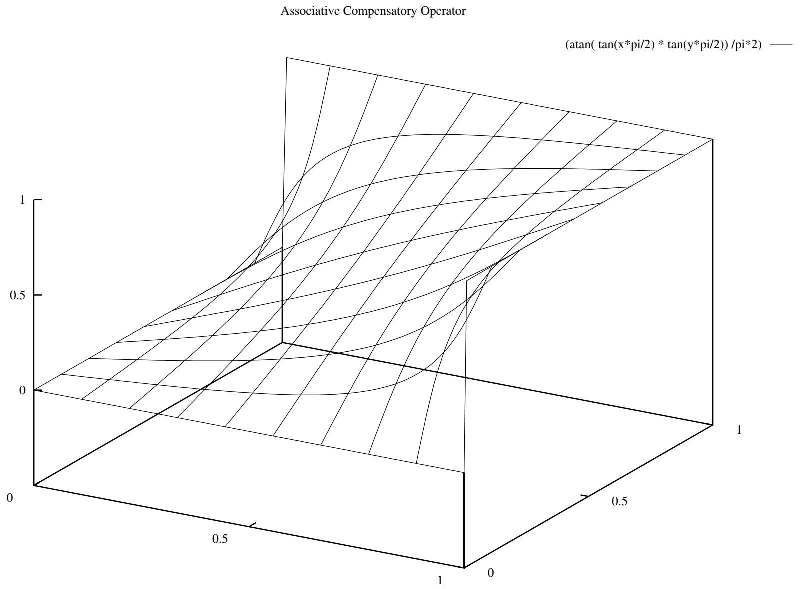

Figure 4: ACOtan - A associative compensatory operator

To create such an operator for FLIP++ with a tangens function as multiplicative generator, we took

$$f(x) = \tan(\pi/2 \cdot x)$$

and

$$f^{-1}(z) = \arctan(z) \cdot 2/\pi.$$

The functionality of the fuzzy aggregation operator ACOtan can be seen in Figure 4.

5 Combination of already represented knowledge

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

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 an other. Or all are 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 just 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.

6 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 by 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 in a recursive way. 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 then the uncertainties about the relations are used in the calculations. This is necessary because otherwise we would derive certain facts from uncertain values. 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.

7 Performance

Because FLIP++ is still not optimized for efficient performance, the performance values should just give an idea of the efficiency of the sytem. We tested the creation of three input variables for chemical percentage differences and derived from them three compatibility constraints by using five relations for each connection. These constraints are aggregated to one overall constraint with two rules. We used the above mentioned ACOtan operator for the aggregation.

The overall constraint was evaluated a thousand times. It was run on a SUN Sparc station 1+ and took 17.9 seconds user time. In each run, each single rule is evaluated for each concerned variable. In this fact lies a great opportunity for gaining a much higher performance.

8 Adaptations and further developments

To make this 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 onto the surface of this library since it is written in an object oriented language.

In future, the next step of the project will be to combine this module with a neural net to find better solutions for a given problem.

References

[1] 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.

[2] Wolfgang Slany. Fuzzy scheduling. CD-Technical Report 94/66, Christian Doppler Laboratory for Expert Systems, Technical University of Vienna, 1994. URL: ftp://mira.dbai.tuwien.ac.at/pub/slany/cd-tr9466.ps.Z.

[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.