FLIP++
A fuzzy logic inference processor library

Markus Bonner, Stefan Mayer, Andreas Raggl, and 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/staff/slany.html#sflip

Abstract

FLIP++ is a fuzzy logic inference processor library. This C++ library was developed to perform calculations with uncertain data and priorities. It evaluates various types of rules, having different importances, 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 a list of membership functions, and second a crisp value 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.

Keywords

Fuzzy control; fuzzy constraint satisfaction problems; compromising; fuzzy qualitative modeling; fuzzy multiple criteria decision making; steelmaking application; C++ library

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

1 Introduction

FLIP++ (first introduced in [1]) is a fuzzy control library together with its graphical user interface from the InterFLIP++ library. Additionally it can be linked together with any other software where fuzzy calculations are needed.

Moreover, FLIP++ is part of the *FLIP++ library [4, 5] 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.

1.1 FLIP++ as part of a scheduling project

*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 genetic algorithm heuristic.
- CheckFLIP++: the knowledge-change consistency checker library that also allows to fine-tune the configuration parameters of a problem.
- InterFLIP++: the graphical user interface 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.
- ReaFLIP++: the reactive optimizer as an extension for DomFLIP++.
- NeuroFLIP++: the learning library for FLIP++ using neural nets.
- TestFLIP++: the version control and test environment for the complete library set.
- SimFLIP++: the simulation toolkit library.

1.2 Organization of this paper

First, we give an overview of how to represent knowledge both qualitatively and quantitatively. Then we describe the relationships within the qualitative representation of knowledge using an example in the steelmaking process. In this section we also emphasize the concept of modularity in complex systems. Furthermore, we show which operators FLIP++ applies and what purpose they are used for. Here we also introduce the associative compensatory operator ACOtan. How to combine already represented knowledge follows in the next section. Afterwards we give a short description of the reasoning process. Finally, we summarize the key points of the paper.

2 Representation of knowledge

A fundamental concept of fuzzy systems is the translation of qualitative knowledge, e.g. represented in human language-like terms, into a machine-usable quantitative representation.

2.1 Qualitative 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 a representation of knowledge is possible. A second advantage is that, like in the human usage of such terms, the exact meaning and 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 made later, or the context can be adapted to new circumstances. Therefore, such qualitative variables do not have 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. Residuals of chemical elements from previous loads influence the quality of subsequent loads to various degrees, depending on the specific chemical element. An overall chemical compatibility constraint assesses to which degree a certain sequence of two loads is desirable. 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 aluminum (Al) as qualitative variable with the allowed terms: *negative*, *zero*, *positive*. Figure 1 shows the corresponding membership functions for such a representation, where term *zero* is selected for graphical editing. We will later derive from this variable the constraint for the compatibility for aluminum by using rules.

2.2 Quantitative representation

For the evaluation of rules that contain qualitative variables, we assign a quantitative representation to the used terms so that ordinary calculation can be performed on them. Each term is described by a membership function which states the membership of this term for each crisp value in a range of possible

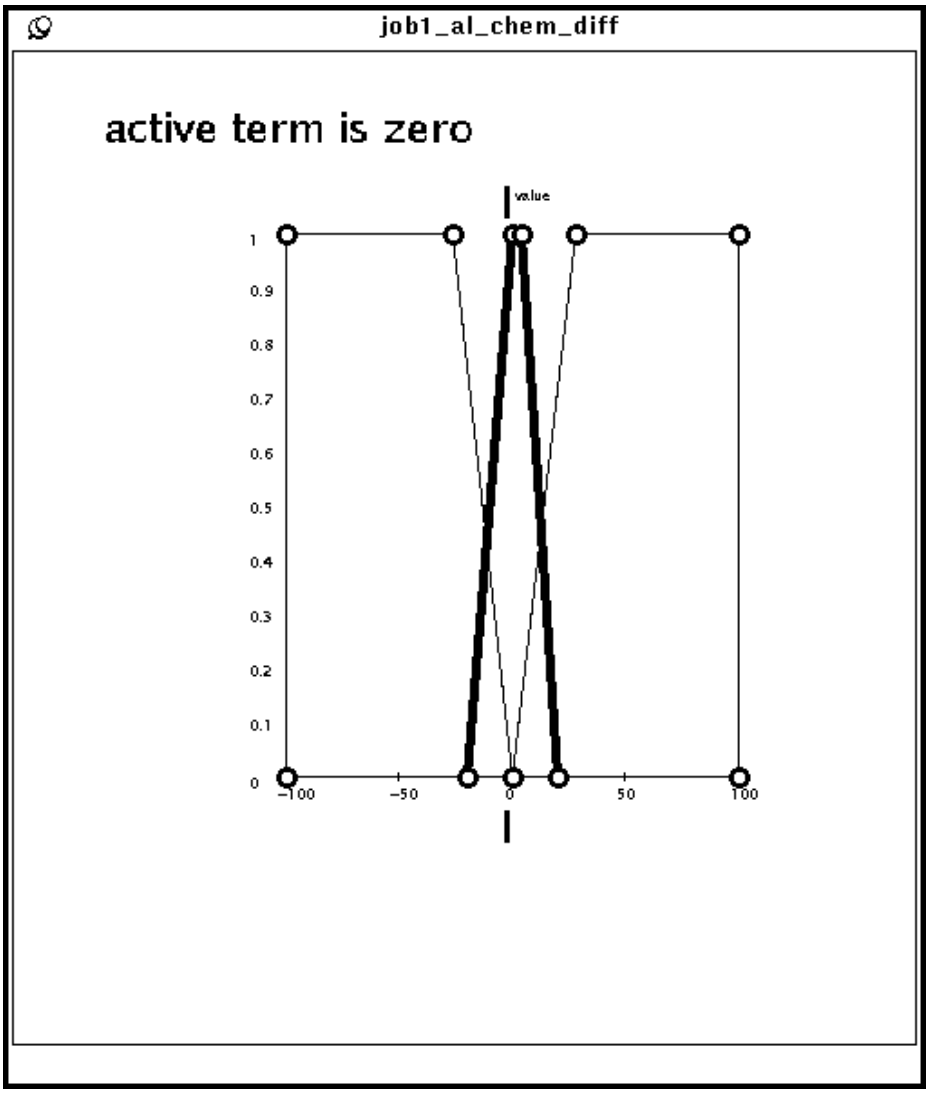

Fig. 1. Membership function for percentage difference of aluminum

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 0 and 1 express a partial membership of the crisp value to the term.

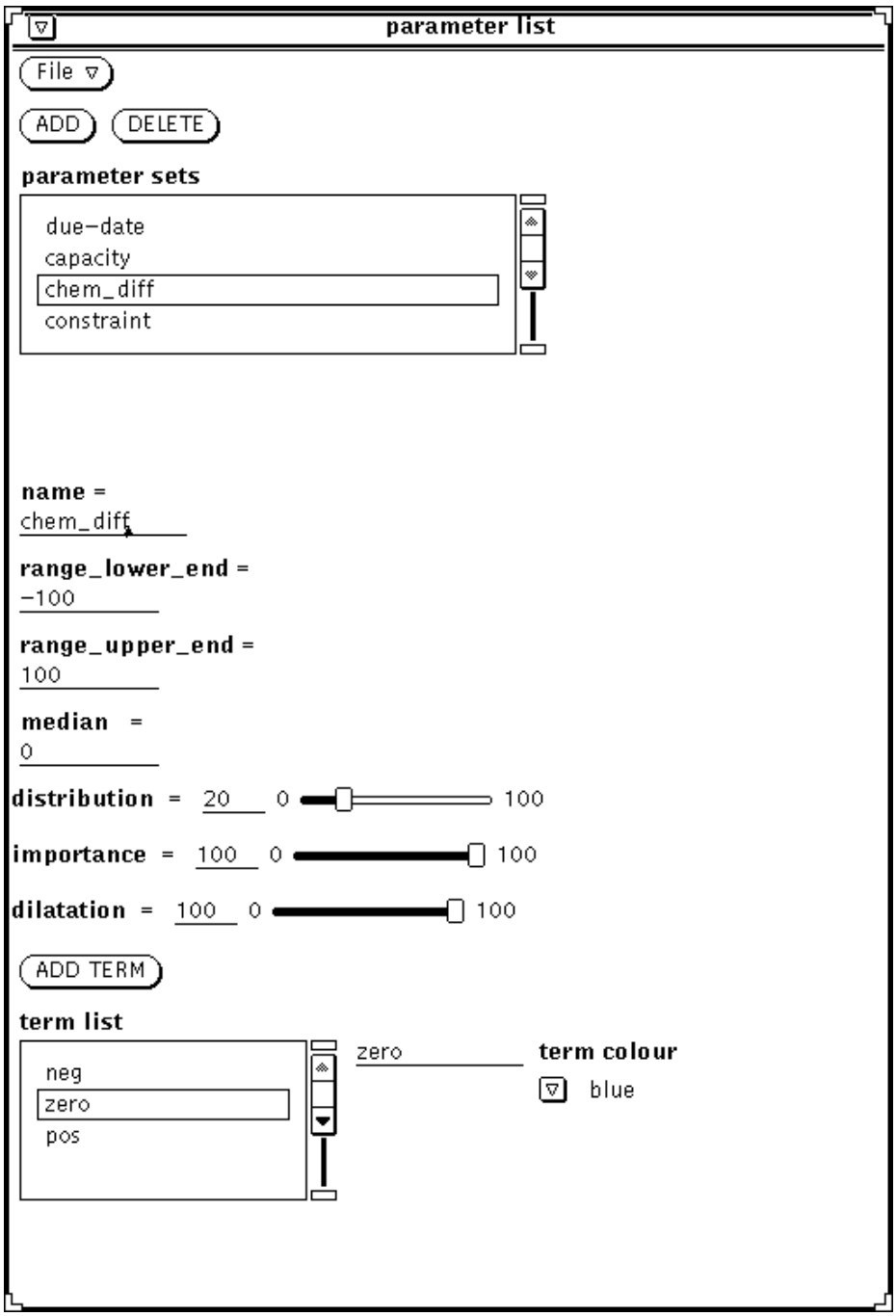

Fig. 2. Set of parameters for membership functions in FLIP++

2.3 Parameter sets

The parameters that define a qualitative variable are shown in Figure 2. Parameter sets are used to ease the generation of variables by defining the number of terms, naming the terms, and determining parameters which are used to shape and position the membership functions. The figure shows the parameter set that can be used for variables representing chemical differences.

A parameter set generates a flexible structure. The membership functions can be directly manipulated according to the specific needs associated with the variable within the graphical user interface as shown in Figure 1.

One idea behind this concept of parameter sets is that a parameter set provides a basis for one type of variables. Such a default variable can then be easily generated to provide a first-hand impression before integrating it into the system.

2.4 Instantiation of variables

When fuzzy inference if-then rules are used to evaluate some given knowledge, this knowledge is contained in variables. The variables on the condition part of the rule have to be instantiated to get a result.

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 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 recursively try to find all rules that evaluate this variable.

3 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 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 condition part of such a rule consist of a variable and a term. Each clause is evaluated by determining the degree of membership of the quantitative representation of its term. All results on the condition part are aggregated, and, in a further step, used to generate new knowledge on the conclusion part. The value for the conclusion part is obtained by applying the aggregated degree of membership of the condition part on its quantitative representation.

3.1 The relation in the example

The degree of satisfaction of the requirements for a certain chemical element is represented through a constraint. The aluminum chemical constraint depends in

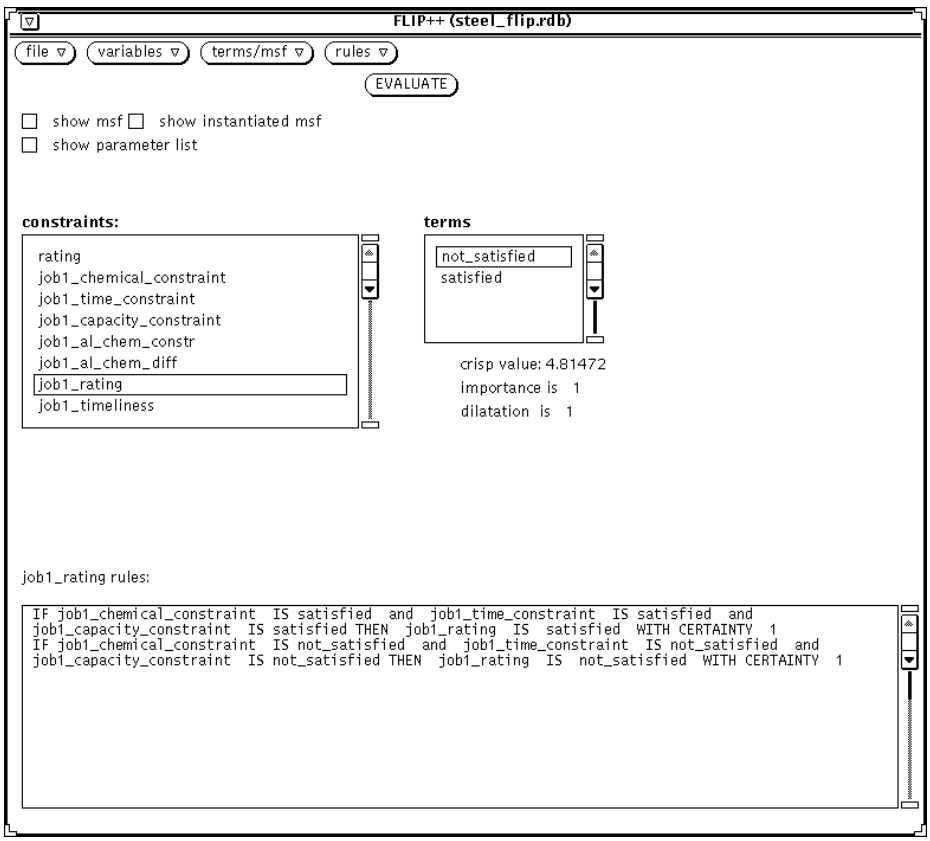

Fig. 3. Graphical user interface with rules and variables

our example on the difference of contents in two consecutive loads, the chemical difference of aluminum. The same scheme is applied for other chemical elements relevant for the steelmaking process. In a further step, the specific chemical compatibility constraints of all chemical elements involved in one job, relative to its preceding job, are the input variables for the global chemical constraint of the same job. Subsequently, we also aggregate all constraints of a job - capacity, chemical, and time - to obtain an overall rating for the job which is exemplified in the rules window of Figure 3. Finally, a global contraint node variable for an overall rating of the production schedule is built by aggregating all individual job ratings.

3.2 Constraint tree - modularity

It is easier and safer to create a complex system of constraints in different modules and then to combine them. A major aim of fuzzy logic is to bring computer systems nearer to reality and human thinking. This makes more sense than try-

Fig. 4. General tree of constraints

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++. The fuzzy conjunction operators of Figure 4 that allow this aggregation are illustrated in Section 4.

4 Operators

In the FLIP++ library we use fuzzy operators for different aspects of the evaluation of given rules and input information. The first place where a fuzzy operator is used, is the application of the importance of a linguistic variable within a rule.

The importance of a variable can be used to differentiate between the linguistic variables at the level of the knowledge. In the next step the single clauses in the condition part of a rule are connected with help of an *and* or *or* operator. Then the certainty of the rule is applied to the result of this action. This is the second level where uncertainty of the given knowledge can be included in the evaluation. After that, the calculated membership degree is used on the linguistic variable of the clause in the conclusion part of a rule. The fuzzy operators we implemented so far in the library are: as t-norm the *minimum* operator, as t-co-norm the *maximum* operator, two parameterizable operaters *fuzzy_and* and *fuzzy_or*, and an associative compensatory operator as described below. Figure 5 shows the operators we have included in the system.

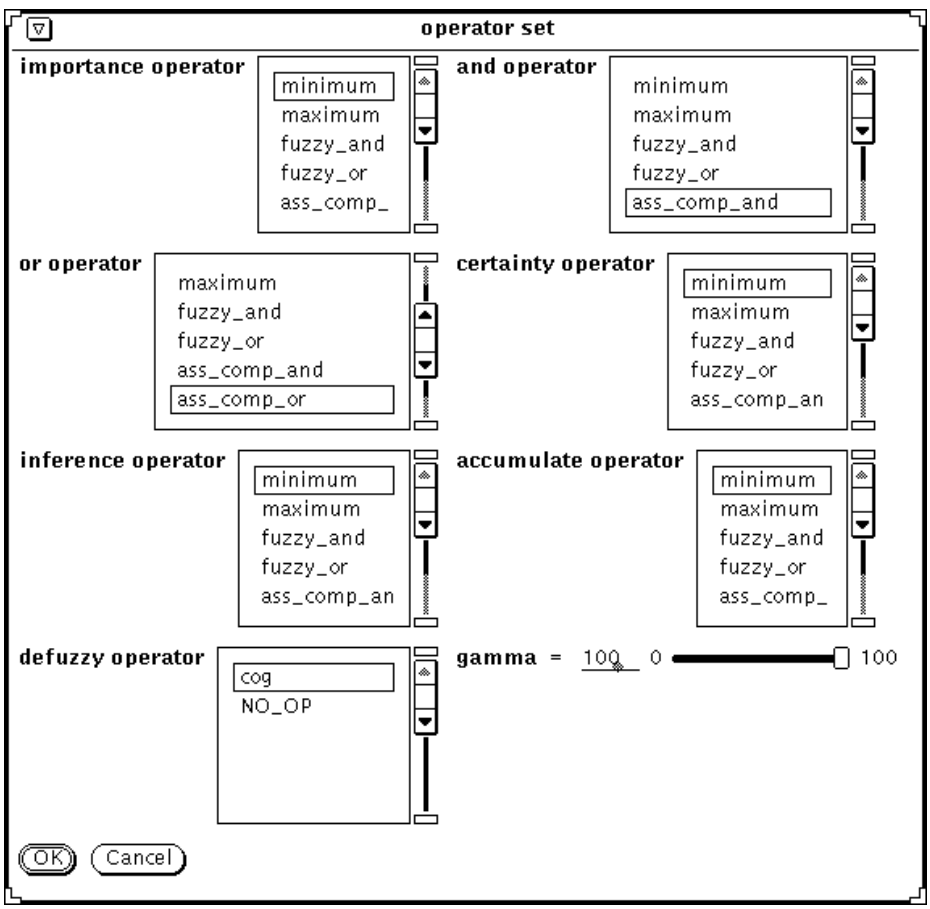

Fig. 5. Default operator set in FLIP++

An important requirement for operators is also derived from the use of hier-

archies mentioned in Section 3.2. Reasoning with such variables and rules creates a need for associativity in the evaluation of rules. Otherwise, the order of single modules in each level of the hierarchy would influence the result.

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. T-norms such as *minimum* fulfill this requirement; and they are often used because of their pleasing mathematical behaviour [2].

A second requirement is that it should represent an aggregation that is similar to the one which a human could use. 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 the logical *and* or *or*. Such aggregations can be done with compensatory operators. The drawback of such compensatory operators is that they normally are not associative [6], and therefore, would not fulfil 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 [3]. From Theorem 1 they derive their Definition 1 for an associative compensatory operator with multiplicative generator f.

Theorem 1 [3] *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 [3] *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.

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

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

and

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

The characteristic plot of this fuzzy aggregation operator ACOtan can be seen in Figure 6.

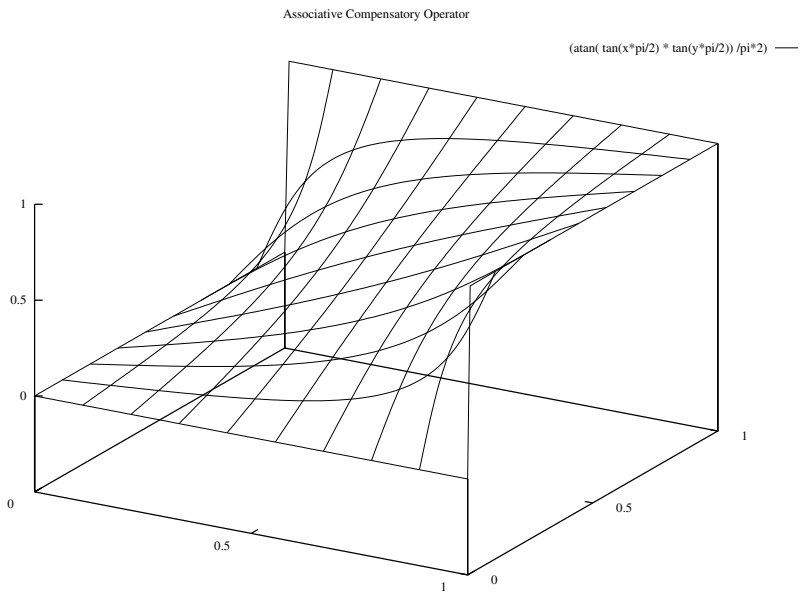

Fig. 6. ACOtan - A associative compensatory operator

5 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 separated 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 built. 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 single modules. The connections between them are then expressed in a new module having new rules, with the output variables as the new inputs. In this case, it is important to evaluate the overall module again after all the changes and new evaluations in lower modules have been done.

6 Reasoning by evaluating knowledge

Reasoning in FLIP++ is done by evaluating the rules and input values whatever their representation is, whether crisp value or instantiated membership function. The rules represent the knowledge about the connections of the qualitative variables, which in turn represent the static knowledge of the domain in form of terms and their quantitative representation. The evaluation is done for one single variable, and all rules that match this one variable are searched among all rules. For all variables used in the conclusion part of such a rule the same is done recursively, so that 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 machinery.

7 Conclusion

To make the FLIP++ library fit better for different applications, there is the possibility of adding new operators. To combine it with other systems one can easily attach a new interface as front end to it since it is clearly structured and well documented.

Now we are implementing an interface to the Stuttgarter neural net simulator to learn the fuzzy rules from existing data. This system will be used in another steelplant application at the VA steel Linz/Austria for desulfurization of steel. Furthermore, within the *FLIP++ library FLIP++ serves also as the basis for a scheduling application to solve shift planning problems.

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 Work-*

shop 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 P. Klement. Operations on fuzzy sets and fuzzy numbers related to triangular norms. In *Proceedings of the Eleventh International Symposium on Multiple-Valued Logic*, pages 218–225, Norman, 1981. IEEE, New York.
3. 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.
4. Wolfgang Slany. Fuzzy scheduling. CD-Technical Report 94/66, Christian Doppler Laboratory for Expert Systems, Technical University of Vienna, 1994. URL: ftp://ftp.dbai.tuwien.ac.at/pub/papers/slany/cd-tr9466.ps.gz.
5. Wolfgang Slany. Scheduling as a fuzzy multiple criteria optimization problem. *Fuzzy Sets and Systems*, 78:197–222, 1996. URL: ftp://mira.dbai.tuwien.ac.at/pub/papers/slany/cd-tr9462.ps.gz.
6. Hans-Jürgen Zimmermann. *Fuzzy Set Theory — and Its Applications*. Kluwer Academic Publishers, 2nd, revised edition, 1991.

This article was processed using the LaTeX macro package with LLNCS style