Scheduling as a fuzzy multiple criteria optimization problem

Wolfgang Slany

Christian Doppler Laboratory for Expert Systems
E184/2, TU Wien, A-1040 Vienna, Austria, Europe
Phone: +43–1–58801–6141 Fax: +43–1–5055304
E-Mail: wsi@vexpert.dbai.tuwien.ac.at

CD–Technical Report 94/62

Labor
Expertensysteme

Christian Doppler Laboratory
for Expert Systems

Technical University of Vienna
Institute for Information Systems
Database and Expert Systems Group

Scheduling as a fuzzy multiple criteria optimization problem

Wolfgang Slany
Christian Doppler Laboratory for Expert Systems
E184/2, TU Wien, A-1040 Vienna, Austria, Europe
Phone: +43–1–58801–6141 Fax: +43–1–5055304
E-Mail: wsi@vexpert.dbai.tuwien.ac.at

April 18, 1994

Abstract: Real-world scheduling is decision making under vague constraints of different importance, often using uncertain data, where compromises between antagonistic criteria are allowed. We explain in theory and by detailed examples a new combination of fuzzy set based constraints and repair based heuristics that help to model these scheduling problems. We simplify the mathematics needed for a method of eliciting the criteria's importances from human experts. We introduce a new consistency test for configuration changes. This test also helps to evaluate the sensitivity to configuration changes. We describe the implementation of these concepts in our fuzzy constraint library ConFLIP++ and in our heuristic repair library DéjàVu. Finally, we present results from scheduling a continuous caster unit in a steel plant.

Keywords: Fuzzy scheduling; fuzzy multiple criteria optimization; fuzzy constraint satisfaction problem; compensation; importance scale; priority; relaxation; decision making; uncertainty management; repair algorithm; tabu list.

Contents

1 Introduction

Scheduling is the task of allocating resources to jobs with respect to time. An example would be production scheduling in industrial environments, where some typical criteria to be optimized might be usage of equipment, space, material and human resources, as well as product quality, production timeliness, minimum total overdue time, minimum number of overdue parts, minimum mean processing time of parts, and robustness to changes due to machine breakdowns. Scheduling has been studied in the operations research literature since the early fifties. Theoretical work has brought several improvements over the years, but there are still many problems with the formal-analytical approaches:

- the algorithms are too complex for real-world applications,
- the models consider only one or two criteria,
- only linear relations among parameters are considered,
- the models demand exact knowledge about durations and technical constraints,
- no antagonistic knowledge can be modeled, and
- the effort to formalize a new scheduling problem is considerable.

A combination of constraint satisfaction techniques and concepts handling vagueness and uncertainty offers a solution to these problems. First, constraint satisfaction techniques can be enhanced by heuristics to reduce the inherent complexity. Second, additional criteria do not necessarily increase the problem complexity when using heuristics, and constraints lend themselves perfectly to model an arbitrary number of criteria. Third, concepts handling vagueness and uncertainty offer the possibility to model non-linear relations intuitively, as well as to reason with incomplete or probabilistic knowledge. We adopted fuzzy set theory and fuzzy logic because they lend themselves well to the task of dealing with vagueness and uncertainty. Fourth, the combination of these techniques can be used to handle contradicting knowledge. And finally, in knowledge based systems, the available knowledge is described explicitly, therefore it is easy to develop and to maintain. Additionally, understandable explanations for decision made can be generated.

The algorithm described in Sections 2 and 3 combines repair based [16, 28] and tabu list [10, 11] techniques with fuzzy constraint relaxation techniques [3, 7, 8, 12, 13, 15, 27]. Its application to flow shop scheduling in a steelmaking environment has been described in [4].

The employed technique is highlighted on a conceptual level in Section 2 from the point of view of fuzzy multiple criteria optimization, as defined in [22, 27]. Constraints from application domains are often vaguely specified and thus lend themselves perfectly for being reformulated as fuzzy constraints. We develop methods of formalizing the relative importance of these constraints. We analyse also the associated problem of handling the chaotic behavior of the search algorithms when parameters describing constraints are changed. Additionally, sensor-measured process variables as well as production parameters are often uncertain, thus calling for the usage of plausible and approximate reasoning, in particular possibilistic logic [6] for constraint evaluation and inference. To facilitate experimentation, a method of softening or hardening complete constraint satisfaction problems is provided. Human experts can thus make trade-offs between for example higher production quality and the ability to schedule more difficult jobs.

Repair based methods as described in Section 3 have proven to be very efficient heuristics for some types of problems with high computational complexity, solving for example the n-queens problem with linear time and space complexity [16]. They are well suited for cooperative and reactive scheduling problems, typical applications in manufacturing. Since unexpected machine failures can happen at any time, the ability to quickly reschedule while changes should be kept as local as possible in order not to disturb unrelated activities is an absolute necessity. Repair based methods are easily transformed into 'anytime' algorithms since good solutions can be found early, and, if time permits, an arbitrary amount of time can be used to find even better ones. The 'anytime' feature is important for reactive scheduling when decisions must be made quickly, time to think is scarce, and settling for a slightly suboptimal solution is acceptable. Since repair based hill climbing is a greedy search method, we combine it with a tabu list technique to avoid being stuck in local maxima or pathological cycles.

2 Fuzzy multiple criteria representation

Consider problems where many different 'criteria' have to be taken into account, various 'objectives', 'aims', or 'goals' must be fulfilled, some 'aspiration levels' should be aimed at, 'domains' of variables must be respected, predefined 'importances' of certain objects have to be considered, briefly, 'side-conditions' have to be observed. On a conceptual level, there is a difference between these notions. For instance, 'criteria' more or less specify what a solution *must* look like, while 'aspirations' specify what a solution *should* look like. However, in an engineering context all these 'side-conditions' are usually formulated in one and the same framework, namely by overloading the term 'constraint' with all these notions. Zimmermann [27] does not distinguish between 'constraints' and 'objectives', arguing that it empirically models the behavior of decision makers quite well. Following this usage, the present text does not differentiate between all these notions.

The theoretical structure needed to deal with problems defined by such constraints will be given in the next sections: Section 2.1 first outlines characteristics of fuzzy constraint satisfaction problems that are of immediate relevance for fuzzy multiple criteria representation issues. Section 2.2 introduces fuzzy constraints formally. Section 2.3 explains aggregation operators fuzzy constraints, including a discussion of aggregation operators that allow trade-offs between several constraints. Section 2.4 goes on to explain how to combine constraints of different importance. Section 2.5 develops ways to compute importance ratio scales for constraints. Section 2.6 introduces a test enforcing consistency between earlier decisions and configuration changes. Section 2.7 presents the tools needed to guide the algorithms explained in Section 3, in particular how to build a decision function taking into account antagonistic constraints. Section 2.8 then describes the fuzzy constraint library ConFLIP++ implemented to test the hypotheses introduced in the previous sections. In Section 3 we blend all these concepts with constraint optimization techniques to tackle real-world scheduling problems.

2.1 Fuzzy constraint satisfaction problems

Constraints are mathematical objects used to make explicit the logic behind a problem. They are used to model decision making problems of e.g. design, planning, or scheduling. Classical constraint satisfaction problems are usually composed of 'crisp' constraints, sometimes called 'boolean', 'binary', 'yes-no', or 'hard' constraints, i.e. relations that can be either satisfied or not, without intermediate state. A solution must satisfy all constraints of the problem. If a problem has more than one solution, it is called an 'underconstrained problem'. To differentiate between these solutions, the decision maker has to consider additional constraints. If a problem has no solution at all, it is called 'overconstrained'. Some constraints must then be 'relaxed' to find acceptable solutions. Both cases are common in real-world situations, for example when buying shoes. Depending on the criteria and objectives considered, there can be many fitting offers, or none at all. Nevertheless, almost everybody will eventually be able to find acceptable shoes, more or less consciously optimizing and compromising between everything that could have some influence on the outcome. Additionally, it turns out that some criteria are easily formulated with words, but cannot as easily be put into a formula. It is for example not easy to formalize how well the shoes will match one's clothes. Human language is also vague when specifying that the new shoes should be *not too expensive*, although there is almost no uncertainty about the price that will have to be paid for a certain pair.

Additionally, fuzzy constraints are well equipped for their use in repair based constraint satisfaction algorithms as discussed in Section 3.1, since they allow to compare satisfaction degrees of constraints in a natural way.

2.2 Fuzzy constraints

A classical crisp k-ary constraint C_{crisp} between a set of variables $x_1, \ldots, x_k \in D_1 \times \ldots \times D_k$, where D_j is the domain of variable x_j, can be formalized as a relation R_{crisp} with its characteristic function $\mathbf{1}_{\text{crisp}}$. R_{crisp} assigns to each k-tuple $(x_1, \ldots, x_k)$ a value $\mathbf{1}_{\text{crisp}}(x_1, \ldots, x_k)$ from $\{0, 1\}$, with the obvious meaning that those k-tuples being assigned 1 are constraint-satisfying 'instantiations' of C_{crisp}, while the others violate the constraint.

In analogy, a soft k-ary constraint C_{soft} between a set of variables $x_1, \ldots, x_k \in D_1 \times \ldots \times D_k$, where D_j is the domain of variable x_j, can be formalized as a relation R_{soft} with its membership function μ_{soft}. R_{soft} assigns a fuzzy membership value $\mu_{\text{soft}}(x_1, \ldots, x_k)$ from $[0, 1]$ to each k-tuple $(x_1, \ldots, x_k)$. The function μ_{soft} represents the level of preference between different instantiations. A value of 1 means that $(x_1, \ldots, x_k)$ fully satisfies R_{soft}. A value of 0 means that $(x_1, \ldots, x_k)$ is incompatible with R_{soft}, i.e. corresponds to a constraint *violation*. An intermediate value means that the corresponding k-tuple *partially* satisfies the constraint. More generally, $\mu_{\text{soft}}(x_1, \ldots, x_k)$ can be interpreted as the degree of satisfaction of the soft constraint.

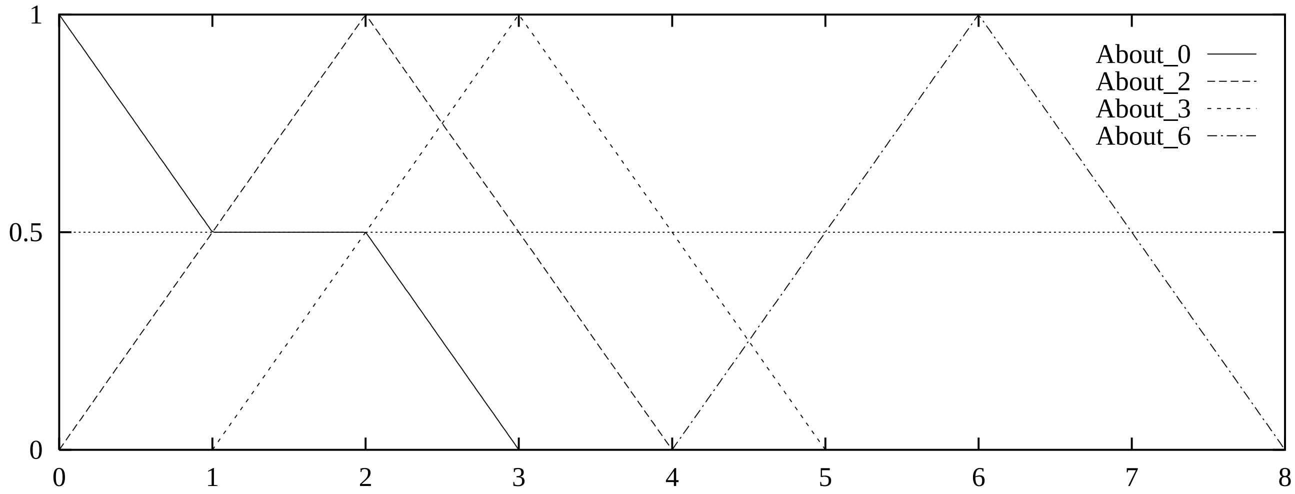

Figure 1: Membership functions of the soft constraints from (2). For instance, 'About_2' is a 'fuzzy number' such that $x_2 = 1.5$ will be a member of it to the degree 0.75. However, the 'tutorial' example as posed in (1) restricts variables to integers.

It is important to note that if $\mu_{\text{soft}}(x_1, \ldots, x_k) = 0$, then the k-tuple $(x_1, \ldots, x_k)$ does not satisfy relation R_{soft} *at all*, i.e. it is a forbidden k-tuple. This allows us to specify 'hard barriers' that should never be crossed when relaxing a soft constraint. Soft constraints *without* hard barriers can easily be specified, too. The membership functions μ of such a soft constraint without hard barriers must simply be defined such that μ never reachs zero, though it can approach zero up to any coefficient $\varepsilon > 0$, always indicating that the respective k-tuple is inferior compared to others with larger μ. This definition of soft constraints with hard barriers allows compensation of *partially satisfied* constraints by other constraints being satisfied to a higher degree, while *violated* constraints cannot be counterbalanced by the satisfaction of other constraints. Therefore, it is in accordance with the remarks by Dubois et al. [7] about what can be correctly termed a constraint satisfaction problem.

An example adapted from Fargier et al. [8] helps to illustrate the properties of these soft constraints. A tutorial is to be organized. The constraints specify that two professors will share the work: Prof. A will give the lecture part of the tutorial and can teach 2 to 4 sessions, with 3 being ideal (C_1); Prof. B will give the training part and can teach about 2 sessions, with 1 and 3 being half-acceptable (C_2); the tutorial should ideally be composed of 6 sessions, but 5 or 7 sessions would be still acceptable (C_3); additionally, Prof. A and Prof. B should teach each about the same number of sessions (ideally, exactly the same number, but a difference of 1 or 2 sessions is half-acceptable) (C_4). Additionally, all sessions must have a prespecified unit length, according to the rule "You can teach anything in mathematics — except something that takes longer than 45 minutes." These constraints can be rewritten formally as:

$$\begin{array}{ll} x_1 = \text{number of lecture sessions} & \text{with } D_1 = \mathbb{N} \\ x_2 = \text{number of training sessions} & \text{with } D_2 = \mathbb{N} \end{array} \tag{1}$$

$$\begin{array}{llll llll} R_1: & x_1 & \text{is} & \text{About\_3} & R_3: & x_1 + x_2 & \text{is} & \text{About\_6} \\ R_2: & x_2 & \text{is} & \text{About\_2} & R_4: & |x_1 - x_2| & \text{is} & \text{About\_0} \end{array} \tag{2}$$

$\mathbb{N}$ represents the set of all natural numbers. The membership functions corresponding to the fuzzy subset 'About_i' that describe the relation between values which can be taken by the variables and satisfaction degree of the constraints are depicted in Figure 1. Note that these soft constraints still have hard barriers. For example, $x_1 = 1$ can never be part of a solution for this problem given the membership function 'About_3' of Figure 1, no matter how much we compromise between constraints.

The motivation behind defining soft constraints lies in their ability to measure the satisfaction of constraints. They allow

- in case of underconstrained problems, to specify preferences in the valid domains of variables, thus providing a natural way to grade solutions that would otherwise be all equal,
- in case of overconstrained problems, to specify margins for constraints where they can be relaxed while still yielding acceptable results, without the necessity to trigger any explicit constraint relaxation procedure,

- the natural definition of priorities among constraints. See Section 2.4 for a discussion of prioritized soft constraints,
- the propagation of uncertain values such as unknown durations as possibility distributions, where the values are ranked according to their level of plausibility as described in Dubois and Prade [6] and in Dubois et al. [7], and
- in both underconstrained and overconstrained cases, to compute in a natural way an objective function that considers all constraints relevant for the problem.

However, not all problems can be solved by using soft constraints alone. The next sections will describe how compensatory aggregation operators and priorities of constraints can enhance the ability to represent a problem adequately.

2.3 Aggregating several fuzzy constraints

The next step in solving a general constraint satisfaction problem is to satisfy *several* constraints with one substitution, i.e. one instantiation of all variables satisfying all constraints at the same time. In classical constraint satisfaction problems, the solution is obtained by taking the intersection of all relations, i.e. calculating the conjunction of all constraints:

$$\mathbf{1}_{R_1\cap\ldots\cap R_n}(x_{j_1},\ldots,x_{j_m}) \quad = \quad \bigwedge_{i=1}^{n} \mathbf{1}_{R_i}(x_{i_1},\ldots,x_{i_k}) \tag{3}$$

with $\{x_{j_1},\ldots,x_{j_m}\} = \cap_{i=1}^{n}\{x_{i_1},\ldots,x_{i_k}\}$ and $m = |\cap_{i=1}^{n}\{x_{i_1},\ldots,x_{i_k}\}|$.

For soft constraints, the same formula as (3) can be adapted by choosing a suitable fuzzy conjunction operator. Zadeh proposed to use the *minimum* in his first article about fuzzy set theory [24]. It has been commonly employed since then, most notably in the very influencing article [1] by Bellman and Zadeh, but there exists a plethora of other operators and operator families. In particular, the use of t-norms (triangular norms) as conjunctions has often been advocated because of their pleasing mathematical behavior.

A t-norm is a binary operator $T : [0,1]^2 \longrightarrow [0,1]$ such that for all $a, b, c, d \in [0,1]$:

$$T(a,1) = a \qquad \text{(neutral element)} \tag{4}$$
$$a \leq b \text{ and } c \leq d \Rightarrow T(a,c) \leq T(b,d) \qquad \text{(monotonicity)} \tag{5}$$
$$T(a,b) = T(b,a) \qquad \text{(commutativity)} \tag{6}$$
$$T(T(a,b),c) = T(a,T(b,c)) \qquad \text{(associativity)} \tag{7}$$

Some basic t-norms with $a, b \in [0,1]$ are

$$\text{minimum:} \quad T_M(a,b) \stackrel{\text{def}}{=} \min(a,b) \tag{8}$$
$$\text{algebraic product:} \quad T_P(a,b) \stackrel{\text{def}}{=} a \cdot b \tag{9}$$
$$\text{Łukasiewicz:} \quad T_L(a,b) \stackrel{\text{def}}{=} \max(0, a+b-1) \tag{10}$$
$$\text{drastic product:} \quad T_W(a,b) \stackrel{\text{def}}{=} \begin{cases} \min(a,b) & \text{if } \max(a,b)=1 \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

More examples including parametrized operator families can be found in Zimmermann [26].

What t-norm operators T do for the intersection is done by t-conorm operator S for the union, corresponding to the disjunction in classical logic. Any t-conorm can be generated from its associated t-norm by applying the following equation:

$$S(a,b) \quad = \quad 1 - T(1-a, 1-b) \tag{12}$$

since the complement, corresponding to negation in classical logic, is usually defined for all $a \in [0,1]$ as $co(a) \stackrel{\text{def}}{=} 1-a$.

The minimum operator as well as all t-norms are actually very strict operators to aggregate constraints since the constraint satisfied the least influences the aggregated outcome maximally. It can be shown (e.g. Yager [23]) that for any t-norm T and for all $a, b \in [0,1]$

$$T(a,b) \quad \leq \quad T_M(a,b) \tag{13}$$

This result together with the associativity (7) of t-norms implies that in multiple criteria decision making, the use of a t-norm type 'anding' forbids compensation for one bad constraint satisfaction. Similarly, the use of pure t-conorm type 'oring' allows for no distraction for one good satisfaction. In both t-norm and t-conorm cases, an indifference to individual criteria versus an over-submission to extreme criteria is shown.

On the other hand, human aggregation procedures in decision environments have been analysed by Zimmermann [26] and it has been shown that humans are able to perform trade-offs between conflicting goals when compensation is permitted. In multiple criteria optimization, this is often a requested feature, and various solutions have been developed. Yager proposes in [23] another type of operator called an ordered weight averaging (OWA) operator. The general definition given for this n-ary operator is restated here, with $a_1, \ldots, a_n, W_1, \ldots, W_n \in [0,1]$, $\sum_i W_i = 1$, and b_j is the jth largest element in the collection $a_1, \ldots, a_n$:

$$\mathrm{OWA}_W(a_1, \ldots, a_n) \stackrel{\mathrm{def}}{=} \sum_{i=1}^{n} W_i \cdot b_i \tag{14}$$

Yager observes that for any t-norm T and any t-conorm S:

$$\underbrace{T_W \le T \le T_M}_{\text{and-type}} \le \underbrace{\mathrm{OWA}}_{\text{and/or-type}} \le \underbrace{S_M \le S \le S_W}_{\text{or-type}} \tag{15}$$

where S_W denotes the t-conorm associated to T_W according to (12).

Nice features of this operator family are that all OWA operators are monotonic, invariant with respect to permutations of the input parameters (corresponds to commutativity in the binary case), and idempotent ($\mathrm{OWA}(a, \ldots, a) = a$), but not associative. However, the latter is usually not required since the OWA operator can be defined for an arbitrary number of parameters, and that is exactly what associativity is normally needed for. Problems can arise when the number of arguments is not known in advance and recursive function calling is needed, e.g. when the number of constraints to be considered is unknown. This is acceptable when one bears in mind that the aggregation of already aggregated values is different from the aggregation of the same values taken directly. To ensure that the hard barrier introduced in Section 2.2 is never crossed when constraints are aggregated, it is necessary that either the aggregation operator propagates a score of zero, or that the scores evaluating to zero get filtered out during the calculation process before aggregation is applied. t-norms propagate zero scores, but it is easy to extend the OWA operator to yield this property too by redefining (14) to:

$$\mathrm{hard\_barrier\_OWA}_W(a_1, \ldots, a_n) \stackrel{\mathrm{def}}{=} \begin{cases} 0 & \text{if } \exists\, i \in [1,n] \text{ such that } a_i = 0 \\ \sum_{i=1}^{n} W_i \cdot b_i & \text{otherwise} \end{cases} \tag{16}$$

Yager notes that the minimum operator is the *only* t-norm having the idempotency property, while all OWA operators have it. This property nevertheless is almost required for multiple criteria optimization. It makes sure that when all criteria are equally satisfied, the overall score is identical to this individual score of all criteria, independently from the number of constraints aggregated.

Interesting weight vectors for multiple criteria optimization problems are vectors that make trade-offs between conflicting goals when compensation is permitted. Consider the following n-ary 'soft and'-like compensation weight vector and its associated OWA operator:

$$W_{\mathrm{soft\_and\_}i} \stackrel{\mathrm{def}}{=} \frac{2i}{n(n+1)} \tag{17}$$

For instance, $n = 4$ results in $\vec{W}_{\mathrm{soft\_and\_4}} \stackrel{\mathrm{def}}{=} (0.1, 0.2, 0.3, 0.4)^\top$.

Actually, the OWA operator family is even more fascinating than can be demonstrated here. It permits to aggregate criteria under the guidance of a quantifier such as "more than four of the criteria must be satisfied." See [23] for a more profound discussion of the properties of the OWA operators, including a worthwhile discussion of their 'andness', 'orness', and 'dispersion'.

The operators introduced are illustrated by extending the tutorial example specified in (1) and (2). It is clear that there is no solution perfectly satisfying all constraints. However, there are several solutions that

	T_W	T_L	T_P	T_M	$\mathrm{OWA}_{\mathrm{soft\_and\_4}}$	S_M
$Sat(\vec{u}_1)$	0	0	0.0625	0.5	0.5	0.5
$Sat(\vec{u}_2)$	0	0	0.25	0.5	0.65	1
$Sat(\vec{u}_3)$	0.5	0.5	0.5	0.5	0.8	1
$Sat(\vec{u}_4)$	0	0	0.25	0.5	0.65	1
$Sat(\vec{u}_5)$	0	0	0.0625	0.5	0.5	0.5

Table 1: Comparing aggregation operators for the tutorial example. The numbers to the right of $Sat(\vec{u}_j)$ are the aggregated evaluation scores computed with the respective operator heading the corresponding column.

partially satisfy the constraints:

$$\begin{array}{lll}
\quad\;\; x_1 \;\; x_2 & & \qquad\qquad\qquad\qquad\;\; R_1 \;\; R_2 \;\; R_3 \;\; R_4 \\
\vec{u}_1 = (\;2,\;3)^\top & \text{with} & Sat_{(R_1,R_2,R_3,R_4)}(\vec{u}_1) = (\;0.5, 0.5, 0.5, 0.5) \\
\vec{u}_2 = (\;3,\;2)^\top & \text{with} & Sat_{(R_1,R_2,R_3,R_4)}(\vec{u}_2) = (\;\;1,\;\;1, 0.5, 0.5) \\
\vec{u}_3 = (\;3,\;3)^\top & \text{with} & Sat_{(R_1,R_2,R_3,R_4)}(\vec{u}_3) = (\;\;1, 0.5,\;\;1,\;\;1) \\
\vec{u}_4 = (\;4,\;2)^\top & \text{with} & Sat_{(R_1,R_2,R_3,R_4)}(\vec{u}_4) = (\;0.5,\;\;1,\;\;1, 0.5) \\
\vec{u}_5 = (\;4,\;3)^\top & \text{with} & Sat_{(R_1,R_2,R_3,R_4)}(\vec{u}_5) = (\;0.5, 0.5, 0.5, 0.5)
\end{array} \tag{18}$$

where $Sat_{(R_1,R_2,R_3,R_4)}$ denotes the vector of the satisfaction degrees for the relations before they are aggregated to compute the actual satisfaction degree.

The aggregated constraint satisfaction values for the tutorial example using the introduced aggregation operators are given in Table 1. As one immediately notices, no useful information can be drawn from the results computed by using T_M. T_M cannot differentiate between the results since it considers only the extreme case, because it satisfies (4), and because it is idempotent as pointed out before. However, it is reasonable to further discriminate between the results since they partially violate a differing number of the constraints, e.g. $\vec{u}_3$ partially violates only R_2, while $\vec{u}_1$ partially violates all constraints. The results using the T_W, the T_L, or the S_M operators are also not very helpful in disambiguating the ranking of the solutions.

T_P and the OWA operator correctly order the solutions with respect to the information available. They indicate that $\vec{u}_3$ is preferable to $\vec{u}_2$ and $\vec{u}_4$, which are preferable to $\vec{u}_1$ and $\vec{u}_5$. However, only the OWA operator is more or less indifferent to the size of the problem because of its idempotency property, while being at the same time sensitive to all the individual scores. In contrast, all t-norms besides T_M are not idempotent but are insofar 'pessimistic' as they never increase when an additional satisfaction degree is taken into account, even when that additional satisfaction degree is better than all degrees so far aggregated. The OWA operators behave differently: They will increase when aggregating additional better satisfaction degrees, decrease when aggregating additional worse satisfaction degrees, and stay equal when aggregating additional equal satisfaction degrees. Thus the behavior of the OWA operator family is often better suited for multiple criteria decision making in real-world situations where multistage inference steps are necessary, in order not to dilute knowledge excessively. More aggregation operators have been described in the literature, e.g. Zimmermann [26], but the operators introduced here illustrate well the main issues involved in a typical fuzzy multiple criteria optimization problem. Zimmermann [26, p. 42] correctly points out that the choice of the most appropriate aggregation operator largely depends on the context of the problem one deals with.

All the operators analysed so far are invariant with respect to permutations of the input parameters, i.e. satisfy the generalized commutativity property. This means that all constraints are equally important. Decisions are based only on the set of scores, but the a-priori importance of the constraint responsible for each individual score is not taken into account. Looking at the tutorial example, this means that the aggregated scores of the solution vectors $\vec{u}_2$ and $\vec{u}_4$ will always be equal (see corresponding rows in Table 1), even though the two solution vectors do not satisfy each constraint to the same degrees. In contrast, the aggregated scores of the solution vectors $\vec{u}_1$ and $\vec{u}_5$ will always be the same irrespective of any priorities, since they satisfy all constraints to the same degrees.

The preference ordering that can be concluded for the tutorial example using any of the aggregation operators introduced so far comprises the following three equivalence classes

$$\begin{array}{c}
\{\vec{u}_1, \vec{u}_5\}, \{\vec{u}_2, \vec{u}_4\}, \{\vec{u}_3\} \\
\text{with} \quad Sat(\vec{u}_1|\vec{u}_5) \;\leq\; Sat(\vec{u}_2|\vec{u}_4) \;\leq\; Sat(\vec{u}_3)
\end{array} \tag{19}$$

i.e. $\vec{u}_3$ will normally be selected, not considering any additional information. The notation $Sat(\vec{u}_1|\vec{u}_5)$ means that either $Sat(\vec{u}_1)$ or $Sat(\vec{u}_5)$ can be written instead of $Sat(\vec{u}_1|\vec{u}_5)$.

Fargier et al. [8] reach the result given in (19) using the minimum operator and non-numeric (inclusion-based and lexicographic-based) ordering methods, similar to the methods used in crisp constraint relaxation techniques (Freuder and Wallace [9] give a very understandable overview concerning crisp constraint relaxation techniques). However, Fargier et al.'s methods assume that a large part of the scores assigned to the constraints for the various instantiations are equal, such as the scores 0.5 and 1 in the tutorial example, since these methods otherwise just select the largest minimum of the individual constraint satisfactions. It is doubtful if not impossible that in a real-world multiple criteria optimization problem involving hundreds or thousands of variables, such as introduced in Section 3.2, the condition that most scores belong to a small finite subset of [0,1] holds. In the latter case, these inclusion-based and lexicographic-based ordering methods will therefore not be needed to disambiguate solutions.

2.4 Fuzzy constraints of different importance

Often criteria do not all have the same importance in real-world applications. It is thus reasonable to consider the relative priorities of constraints when instantiations are evaluated and compared. An intuitive requirement is that as a constraint becomes more important, it should play a more significant role in determining the overall decision function.

Following-up the tutorial example, we would like to take into account additional information about preference between constraints. For example, having a tutorial composed of 6 sessions might be *very* important because of the way the tutorial will be paid for. In addition, Prof. B's wishes might be *more* important than observing any other side-conditions since Prof. B is the Dean of the Faculty. Accordingly, the precedence of the constraints from (2) would reduce to:

$$\{R_2\} \overset{\text{prec}}{\succ} \{R_3\} \overset{\text{prec}}{\succ} \{R_1, R_4\} \tag{20}$$

Given this additional information, the five solutions listed in (18) do not obey any more the relation given in (19). $\vec{u}_1$ and $\vec{u}_5$ will remain the least desirable options. $\vec{u}_4$ is certainly preferable to $\vec{u}_2$, since $\vec{u}_4$ better satisfies the *more important* relation R_3, in contrast with $\vec{u}_2$ better satisfying only the *normal* relation R_1, everything else being equal. However, no preference ordering can be asserted concerning how $\vec{u}_3$ should be positioned in relation to $\vec{u}_2$ and $\vec{u}_4$ without further numerical preference information. Any situation will be possible, depending on the relative importance assigned to the different constraints. Formally, the situation summarizes at this stage to

$$Sat(\vec{u}_1|\vec{u}_5) \;\leq\; Sat(\vec{u}_2|\vec{u}_3|\vec{u}_4) \quad \text{and} \quad Sat(\vec{u}_2) \;\leq\; Sat(\vec{u}_4) \tag{21}$$

of which (19) is one particular case.

Several approaches have so far been proposed in the literature to model and further remove the ambiguity remaining in situations such as (21). Yager [22] argues that ranking or weighting of objectives can be achieved by: linear orderings of objectives; intervals; relative ratio; or absolute ratings. These are progressively more difficult to obtain from a human expert. Forcing the latter to provide such information may yield incorrect answers if the expert simply *cannot* give this information accurately. Furthermore, as the assessment scale becomes more refined, it becomes more sensitive to noise and, consequently, more error prone.

Yager [22] proposed a new methodology that enables him to include different importances while requiring only an *ordinal* scale (i.e. a finite set of ordered symbols) for preference information. For a particular objective the negation of its importance acts as a barrier such that all ratings of alternatives that are below that barrier become equal to the value of the barrier. The motivation is that the implication $a \Rightarrow b$ can be interpreted as a fuzzy $\neg a \vee b$, with negation being the fuzzy complement and '$\vee$' being a fuzzy disjunction. That is, Yager disregards all distinctions lower than the barrier while keeping distinctions above the barrier. A detailed example about selecting a car is given in Yager's article [22].

The approach by Fargier et al. [8] is similar to Yager's [22]. Fargier et al. propose to order constraints with respect to each other by giving them a priority degree. A coefficient $w \in [0, 1]$ is attached to each constraint, with a higher w indicating a comparatively higher importance. These priorities are transformed, without any loss of information, into constraint satisfaction degrees:

$$\mu_{\text{soft},w}(x_1, \ldots, x_k) \;\overset{\text{def}}{=}\; \max(1 - w, \mu_{\text{soft}}(x_1, \ldots, x_k)) \tag{22}$$

Fargier et al. [8] motivate their choice by arguing that this formalism ensures that w represents "the degree to which [a constraint] *must* be satisfied [... The number $(1 - w)$] measures to what extent it is possible to violate [a constraint]." Figure 2 visualizes in the plot titled "max(c,not(w))" the relation between the unweighted

(24)	R_1	R_2	R_3	R_4	rank
$\vec{w}_{24}$	1/2	1	3/4	1/2	
$\vec{u}_1$	0.5	0.5	0.5	0.5	III
$\vec{u}_2$	1	1	0.5	0.5	II
$\vec{u}_3$	1	0.5	1	1	I
$\vec{u}_4$	0.5	1	1	0.5	II
$\vec{u}_5$	0.5	0.5	0.5	0.5	III

(25)	R_1	R_2	R_3	R_4	rank
$\vec{w}_{25}$	0	1	1/2	0	
$\vec{u}_1$	1	0.5	0.5	1	III
$\vec{u}_2$	1	1	0.5	1	II
$\vec{u}_3$	1	0.5	1	1	II
$\vec{u}_4$	1	1	1	1	I
$\vec{u}_5$	1	0.5	0.5	1	III

(26)	R_1	R_2	R_3	R_4	rank
$\vec{w}_{26}$	0	1	3/4	1/4	
$\vec{u}_1$	1	0.5	0.5	0.75	IV
$\vec{u}_2$	1	1	0.5	0.75	III
$\vec{u}_3$	1	0.5	1	1	II
$\vec{u}_4$	1	1	1	0.75	I
$\vec{u}_5$	1	0.5	0.5	0.75	IV

(27)	R_1	R_2	R_3	R_4	rank
$\vec{w}_{27}$	0	1	2/5	0	
$\vec{u}_1$	1	0.5	0.6	1	IV
$\vec{u}_2$	1	1	0.6	1	II
$\vec{u}_3$	1	0.5	1	1	III
$\vec{u}_4$	1	1	1	1	I
$\vec{u}_5$	1	0.5	0.6	1	IV

(28)	R_1	R_2	R_3	R_4	rank, operator dependent	
$\vec{w}_{28}$	0.29	1	3/4	0.29	T_P	$\text{OWA}_{\text{soft\_and\_4}}$
$\vec{u}_1$	0.71	0.5	0.5	0.71	IV	IV
$\vec{u}_2$	1	1	0.5	0.71	III	III
$\vec{u}_3$	1	0.5	1	1	II	I
$\vec{u}_4$	0.71	1	1	0.71	I	II
$\vec{u}_5$	0.71	0.5	0.5	0.71	IV	IV

Table 2: Some rankings of solutions with weighted constraints according to (22) depicted in Figure 2 in the plot titled "max(c,not(w))" for the tutorial example. The numbers to the right of $\vec{w}_i$ are the weights associated with the relations from (2). The numbers to the right of the solution vectors $\vec{u}_j$ below the weights are the scores of the weighted constraints. The rankings are based on the aggregated evaluations for the corresponding instantiation vector $\vec{u}_j$, which are independent of the 'and' operator used, with the notable exception of the solutions in (28).

constraint satisfaction $c \stackrel{\text{def}}{=} \mu_{\text{soft}}$, the priority degree w, and the resulting weighted constraint satisfaction score $\mu_{\text{soft},w}$. We observe that for the case $w = 1$, $\mu_{\text{soft},1} = \mu_{\text{soft}}$. This means that $w = 1$ is a neutral priority degree, a reasonable feature shared with other schemes to implement priorities between constraints. Yager's [22] approach is slightly different in that he does not use numbers but a finite set of ordered symbols to represent the weights of the criteria, using the following linear ordering

$$\text{perfect} \geq \text{very high} \geq \text{high} \geq \text{medium} \geq \text{low} \geq \text{very low} \geq \text{lowest} \tag{23}$$

and defining the semantic of the '$\neg$' operator in an obvious way (e.g. $\neg\,\text{high} = \text{low}, \ldots$). The difference between Yager's method and the method specified by Fargier et al. is only of terminological nature when a 'suitable' fine-grained linear ordered set of symbolic priority values and the minimum operator are used. Other aggregation operators may produce intermediate values that are not necessarily among the linear ordered set characteristic for Yager's method, therefore making Fargier et al.'s more universal. This is largely dependent on the problem parameters, for instance how many criteria must be weighted and aggregated. Since the two models are so similar, the tutorial example serves again to exemplify their mechanics by use of the method given by Fargier et al. Results of applying some weight vectors consistent with the ordering given in (20) to the solution vectors from (18) are presented in Table 2.

The ranking shown for instance in (24) is computed through the following steps:

1. The weight vector $\vec{w}_{24} = (1/2, 1, 3/4, 1/2)^\top$ is applied to the respective unweighted solution vector $\vec{u}_j$ from (18). The resulting values are those written in the cells of the array (24), always being associated with one solution vector $\vec{u}_j$-row and one constraint relation R_i-column taken from (2).

2. These weighted evaluations of the individual constraints are aggregated by use of any 'and'-like operator, for instance the $\text{OWA}_{\text{soft\_and\_4}}$ operator as defined in (17). These aggregated values are not included in Table 2 as they depend upon the operator actually used. Since the resulting ranking is — with some important exceptions discussed below — independent of the 'and' operator used, only the result of the next step is included in the cells of the array (24).

3. The aggregated evaluations of each solution vector are used to rank them as indicated in the last column.

All the rankings found in Table 2 are consistent with the ranking relation given in (21), because all weight vectors are consistent with the priority ordering of the constraints as defined in (20). Additionally, all of them definitely resolve the ambiguity that remained in (21). Note that when using the T_M or the T_L operators, further disambiguation with the non-numeric (inclusion-based and lexicographic-based) ordering methods proposed by Fargier et al. [8] may be necessary.

The ranking found in (24) is identical to the one found in Section 2.3, implying that the weights assigned to the constraints could not outweight the advantage of the solution vector $\vec{u}_3$ that satisfied perfectly a larger number of constraints. Note also that solution vector $\vec{u}_3$ can take rank I in (24), rank II in (25), or rank III in (27), depending on the weights used, each ranking being consistent with (21).

Case (28) is particularly interesting as it emphasizes the importance of fine tuning the weights for a particular aggregation operator: Depending on the operator used, solution vector $\vec{u}_3$ and $\vec{u}_4$ switch ranks I and II. The explanation for this somewhat erratic behavior is that the 0.71 score is just between the fixpoints of the aggregation operators for $\vec{u}_3$ and $\vec{u}_4$. A fixpoint of an operator is a value for which the operator will yield the same satisfaction scores for both solution vectors, thus ranking them identical. Since those fixpoints are different ($\sqrt{1/2} \simeq 0.70711$ for T_P versus $5/7 \simeq 0.71429$ for $\mathrm{OWA}_{\mathrm{soft\_and\_4}}$), there is a slim interval where the two operators can specify a different ranking.

The results given in Table 2 show that the method introduced so far is able to represent the importance of constraints adequately. Nevertheless, there are problems with this way to implement constraints of varying importance: For one, $\mu_{\mathrm{soft}} = 0 \not\Rightarrow \mu_{\mathrm{soft},w} = 0$ as the latter is equal to $1 - w$. However, the implication is needed to make sure that the hard barrier introduced in Section 2.2 is never crossed when constraint relaxation occurs. Thus a violated constraint will not propagate through to the evaluation function for a complete instantiation. One solution to avoid crossing the hard barrier is to test it at evaluation time and propagate the test result as an additional boolean flag. Another solution sketched in Dubois et al. [7] and termed 'safeguarding' constraints is to add a crisp constraint (with lower importance) that makes sure that a certain hard barrier is never crossed.

Additionally, using this type of importance characterization just allows to *decrease* importance, but never to *increase* it, as one immediately observes in Figure 2 in the plot titled "max(c,not(w))". This can be shown by the fact that for all $\mu, w \in [0,1], \max(1-w, \mu) \geq \mu$, and remembering that *higher* scores influence the outcome *less* when using any 'and'-like aggregation operator, as it is usually done in multiple criteria decision making. A solution is to first set all constraint priorities to an intermediate value such as 0.5 and then to increment or decrement this value.

The ordering behavior of the priority values is somewhat chaotic, as illustrated by the rankings in Table 2. The rankings seem to be sensitive to certain threshold values for the weights. Since there is no unique way to compute the weights for a given problem, there is also no clear way to relate the weights to the wishes of the field expert regarding priorities, besides experimenting and fine tuning by testing different weight vector variants. Section 2.6 indicates how this tuning can be done rationally while avoiding inconsistencies with former decisions.

Another priority-scheme that remedies the problems just mentioned is analysed in the following paragraphs. Yager suggested to use the minimum operator for aggregation and to use formula

$$\mu_{\mathrm{soft}}^{\mathrm{Yager},w}(x_1,\ldots,x_k) \stackrel{\mathrm{def}}{=} (\mu_{\mathrm{soft}}(x_1,\ldots,x_k))^w \tag{29}$$

instead of the one given in (22), with $w \in [0,1]$, to account for importance of a constraint. Yager notes that the importance measures can be expressed on the unit interval because of

$$\min(x_1^{kw_1}, x_2^{kw_2}) = (\min(x_1^{w_1}, x_2^{w_2}))^k \tag{30}$$

with $kw_1, kw_2 \in \mathbb{R}^+$ and $w_1, w_2 \in [0,1]$. However, Equation 30 is only true for the minimum operator, but it does not universally hold for other t-norms nor for the OWA operator family. Additionally, the case $\mu_{\mathrm{soft}} = 0$ and $w = 0$ must be considered in the light of the deliberations regarding the hard barrier issue raised in Section 2.2. Setting $\mu_{\mathrm{soft}}^w(x_1,\ldots,x_k) \stackrel{\mathrm{def}}{=} 1$ when $\mu_{\mathrm{soft}} = 0$ and $w = 0$ would specify that a constraint with weight zero is 'turned off'. This constraint would further have no relevance for the decision process, even when it is totally violated. Therefore, (29) is redefined for the general case and $w \in \mathbb{R}^+$ such that:

$$\mu_{\mathrm{soft}}^w(x_1,\ldots,x_k) \stackrel{\mathrm{def}}{=} \begin{cases} 1 & \text{if} \quad \mu_{\mathrm{soft}} = 0 \text{ and } w = 0 \\ (\mu_{\mathrm{soft}}(x_1,\ldots,x_k))^w & \text{otherwise} \end{cases} \tag{31}$$

Figure 2 visualizes in the plots titled "Yager" the relation between the unweighted constraint satisfaction

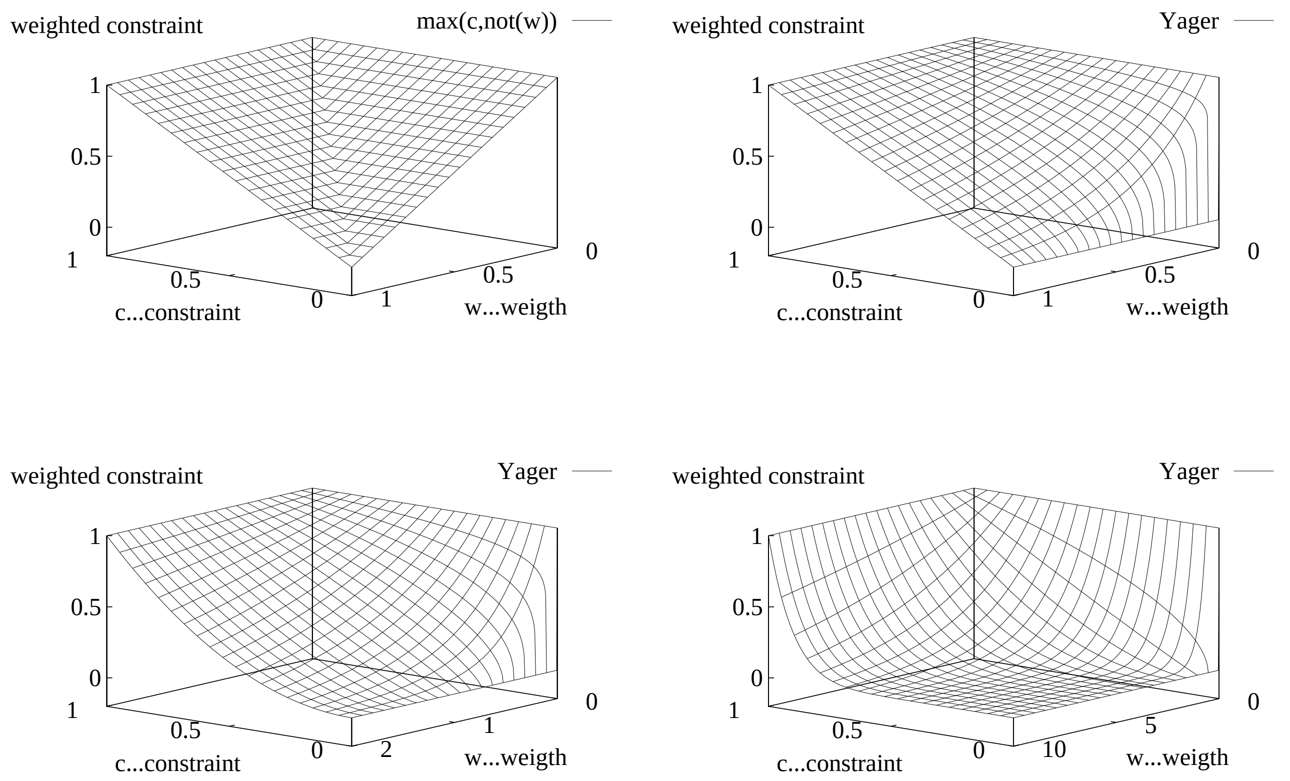

Figure 2: Satisfaction taking into account priority according to Definition (22) in the plot titled "max(c,not(w))", and according to Definition (31) in the plots titled "Yager", with $w \in [0,1]$, $w \in [0,2]$, and $w \in [0,10]$. The methods are respectively applied in Table 2 and in Table 4 to the tutorial example.

$c \stackrel{\text{def}}{=} \mu_{\text{soft}}$, the priority degree w, and the resulting weighted constraint satisfaction score μ^w_{soft} for $w \in [0,1]$, $w \in [0,2]$, and $w \in [0,10]$. Note that μ^w_{soft} is not differentiable at the point $\mu_{\text{soft}} = 0$ and $w = 0$ (because 0^0 is undefined), but is artificially set to 1 for that point; it is 1 for all points on the line $w = 0$, though it is 0 for all other points on the line $\mu_{\text{soft}} = 0$. This is a natural behavior: One can expect μ_{soft} to change more often than w, because w is usually set for a particular constraint for as long as one believes that the constraint should have that particular importance, i.e. often for the whole lifetime of the constraint. Setting the priority of a constraint to zero implies that it should be turned off, i.e. neglected for the decision making process. If an OWA-type operator is used for aggregation, the constraints having weight zero should not be included in the aggregation process since they otherwise influence the outcome inappropriately. In all other cases, $\mu_{\text{soft}} = 0$ should imply that the corresponding constraint is 100% violated and the corresponding instantiation does not belong to the solutions of the soft constraint satisfaction problem.

Another property worth noting is that μ^w_{soft} is linear and equal to μ_{soft} for $w = 1$, which means that weight 1 is a neutral importance value, as one observes on the plot with $w \in [0,1]$ titled "Yager" in Figure 2. Additionally, for all $w < 1$, $\mu^w_{\text{soft}} > \mu_{\text{soft}}$. A smaller importance factor than 1 increases the membership function of the prioritized constraint and therefore makes it less constraining for 'and'-like operator based decisions. Conversely, for all $w > 1$, $\mu^w_{\text{soft}} < \mu_{\text{soft}}$. A larger importance factor than 1 decreases the membership function of the prioritized constraint and therefore makes it more constraining for 'and'-like operator based decisions. In the next section, the μ^w_{soft} weighting scheme is applied to the tutorial example.

2.5 How to find the importance of constraints?

We demonstrated in the previous section the usefulness of importance factors to model multi criteria optimization problems. The question nevertheless remains how to correctly ascertain the importance factors on a ratio scale. A procedure developed by Saaty [17] and restated by Ibrahim and Ayyub in [14] can be applied to compute

relative importance	human-readable definition
1	equal importance
3	weak importance of one over another
5	essential or strong importance
7	very strong or demonstrated importance
9	absolute importance
2, 4, 6, 8	intermediate values between two adjacent scale judgments

Table 3: Relative importance attributes proposed by Saaty [17]. A human expert can specify weights between constraints, using the informal descriptions in the column on the right.

the absolute importance factors for each constraint from paired comparisons between constraints. When the absolute importance factors for m constraints should be computed, these m constraints are pairwise compared with each other, their relative importance being written in an m-by-m matrix

$$A = \begin{pmatrix} 1 & a_{12} & \cdots & a_{1m} \\ a_{21} & 1 & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & 1 \end{pmatrix} \qquad (32)$$

where a decision maker judges that constraint C_i is a_{ij} times more important than constraint C_j. In order to guarantee that matrix A is 'self-consistent', only $(m-1)$ 'logically independent' pairwise comparison statements are collected to construct it. The rest of the values follows by applying

$$\forall\, i,j,k \in [1,m]:\ a_{ij} = 1/a_{ji}\ \ (\text{therefore}\ \ a_{ii} = 1)\ \ \text{and}\ \ a_{ij} = a_{ik}a_{kj} \qquad (33)$$

The term 'logically independent' means that it should not be possible to compute any of the comparison statements given by the decision maker through the construction rules from (33) applied to the other comparison statements given by the decision maker. That is, the graph, with node-set $\{1,\ldots,m\}$ and edges (i,j) for every comparison of C_i with C_j, should be a connected tree with edgeweights a_{ij}. The full matrix A can then be defined by calculating each a_{ij} as the product of the weigths along the unique path between nodes i and j in this tree.

The eigenspace $E_{\max} = r(e_1,\ldots,e_m)^\top$, with $r \in \mathbb{R}$, corresponding to the maximum eigenvalue $\lambda_{\max}$ of A, 'normalized' to $\widehat{E}_{\max} = (\widehat{e}_1,\ldots,\widehat{e}_m)^\top$ such that $\sum_{k=1}^{m} \widehat{e}_k = 1$, and finally multiplied with m is then the searched absolute weight vector $\vec{w}$ for the constraints:

$$\vec{w} = m\frac{E_{\max}}{\sum_{k=1}^{m} e_k} \qquad (34)$$

The weighting makes sure that when all constraints are equally important, they are all weighted with 1, which means that $\mu^w_{\text{soft}} = \mu_{\text{soft}}$. A simplification following from the construction rules (33) of a self-consistent matrix A is that $\lambda_{\max} = m$. To calculate the corresponding eigenspace $E_{\max}$, one should solve the equation

$$AE_{\max} = \lambda_{\max}E_{\max} \qquad (35)$$

Actually, another simplification following from the construction rules (33) of a self-consistent matrix A is that any column of matrix A generates $E_{\max}$. Therefore, $E_{\max}$ has not actually to be computed by any algorithm such as the Gaussian elimination procedure. This becomes totally clear when interpreting the elements of the jth-column as "how many times more important constraint C_i (of row i) is compared to C_j," as defined above, so that the jth column vector is a_{1j} times the first column vector. Therefore, the whole procedure can be simplified considerably compared to the one given in [14] since Equation (35) can be completely neglected. It is only necessary to construct a self-consistent matrix A as explained above and then to compute $\vec{w}$ out of e.g. the first column of A. Instead of constructing A completely, it also would suffice to identify one least important constraint C_k, and then to specify the relative importance a_{ik} of the others compared to C_k.

Saaty [17] proposes to use importance attributes as described in Table 3, "due to the human ability to make effective quantitative distinctions [only between] five attributes [...] Compromises between attributes can

(36)	R_1	R_2	R_3	R_4	rank
$\vec{w}_{36}$	1/4	9/4	5/4	1/4	
$\vec{u}_1$	0.84	0.21	0.42	0.84	IV
$\vec{u}_2$	1	1	0.42	0.84	II
$\vec{u}_3$	1	0.21	1	1	III
$\vec{u}_4$	0.84	1	1	0.84	I
$\vec{u}_5$	0.84	0.21	0.42	0.84	IV

(37)	R_1	R_2	R_3	R_4	rank, operator dependent	
$\vec{w}_{37}$	8/11	16/11	12/11	8/11	T_M	$\text{OWA}_{\text{soft\_and\_4}}$
$\vec{u}_1$	0.60	0.36	0.47	0.60	IV	IV
$\vec{u}_2$	1	1	0.47	0.60	II	III
$\vec{u}_3$	1	0.36	1	1	III	I
$\vec{u}_4$	0.60	1	1	0.60	I	II
$\vec{u}_5$	0.60	0.36	0.47	0.60	IV	IV

Table 4: Some rankings of solutions with weighted constraints according to (31) depicted in Figure 2 in the plots titled "Yager" for the tutorial example. The numbers to the right of $\vec{w}_i$ are the weights associated with the relations from (2). The numbers to the right of the solution vectors $\vec{u}_j$ below the weights are the scores of the weighted constraints. The rankings are based on the aggregated evaluations for the corresponding instantiation vector $\vec{u}_j$.

be used where greater precision is needed." Because $\vec{w}$ is the product of a normalized eigenvector of A and the number of constraints, it is relatively independent of the ad-hoc numbers given in Table 3.

The tutorial example illustrates the details of the procedure of Saaty [17]. Assume for instance that according to a specialist, the weights are assigned to the constraints established in (2) using Table 3 such that C_2 is absolutely (9 times) more important than C_1; C_3 is essentially (5 times) more important than C_1; and C_4 is equally (1 time 'more') important as C_1. This gives us the first column of matrix A, which is enough to set $E_{\max} = (r, 9r, 5r, r)^\top$, with $r \in \mathbb{R}$, therefore $\widehat{E}_{\max} = (1/16, 9/16, 5/16, 1/16)^\top$ and $\vec{w} = (1/4, 9/4, 5/4, 1/4)^\top$. Applied to the solutions of (18), the values given in (36) are computed using these weights and the weighting scheme from (31), and displayed in Table 4 up to two radix places. The resulting ranking corresponds well with the intuitive priorities assigned to the constraints using the values from Table 3. Note that when using t-norms for aggregation, further disambiguation with the non-numeric (inclusion-based and lexicographic-based) ordering methods proposed by Fargier et al. [8] may be necessary, as before. The $\text{OWA}_{\text{soft\_and}}$ operators immediately compute the correct ordering.

The problems that existed with the weighting scheme of (22), namely that the hard barrier could be crossed, that importance could not be increased, and that the ordering behavior was not smooth, are solved by the weighting scheme proposed in (31). Nevertheless, as the example of (37) illustrates, weights make sense only in combination with a previously-known aggregation operator. The weight vector $\vec{w}_{37}$ in Table 4 was computed by normalizing and then multiplying with $m = 4$ the weight vector $\vec{w}_{24}$ from Table 2, to bring it into accordance with the weighting scheme proposed by Saaty [17]. The examples (37) and (24), as well as the example (28) demonstrate that fine tuning of the weights for a prespecified $\langle$aggregation-operator, weighting-scheme$\rangle$ combination is absolutely necessary in order to obtain meaningful results. It is possible to completely change the ranking behavior of weights by switching to another aggregation operator or to a different weighting scheme.

2.6 A consistency test for configuration changes

An answer to the problem of making sure that fine tuning is done consistently with earlier decisions is to adopt a consistency test for configuration changes. Such configuration changes could be changes in the priorities between constraints, adopting a new aggregation operator, changing hard barriers, changing membership functions, or changing the logical structure of constraints. Basically, this change together with the test produces a new ranking for a given set of new instantiations, while observing predefined rankings for a set of old reference ranking of pairs of instantiations. The mechanism works such that, if the human expert is dissatisfied with a ranking produced by the system, the human expert can slightly change the weights of some constraints, or the exact form of some membership function (e.g. to specify that the hard barrier is actually located slightly higher), or any other parameter of the problem, such as the aggregation operator used. A consistency test will then check whether the new configuration is consistent with the rankings for a set of reference pairs of instantiations. This is done by applying the new configuration, e.g. the set of new weights, to all the old ordered pairs of instantiations, and calculating their evaluation scores with this new configuration. If for each reference pair the order between the two reference instantiations remains unchanged, this indicates that the new configuration does not invalidate any previous reference ordering. It is compatible with all decisions made in the past that became reference ranking pairs.

If one reference ranking pair is ranked in the opposite order, this means that either the new configuration is *wrong* and has to be changed again, or that some reference ranking pairs are obsolete and should therefore be removed from the reference ranking pair database. In both cases, an inconsistency among the reference rankings and the new ranking is pointed out. This inconsistency has to be resolved such that the resulting system makes rational, predictable, understandable and self-consistent decisions. Whether such a decision making behavior can be termed *objective* or *subjective* depends on other factors. However, it is usually possible to lead *several* human experts to agree on a common, undisputed subset of some reference ranking pairs of instantiations, or at least to establish several different sets that correspond to configurations which can be further characterized by (and saved for later reuse under) such names as 'risky/cost-cutting', 'highest-quality', 'observe-temporal-constraints', 'standard-mix', etc., indicating their general tendency for decision making. The corresponding last configuration is saved together with these reference ranking pairs of instantiations as one knowledge base. Of course not all intermediate stages have to be stored permanently. This permits modeling with maximal flexibility the intentions of the human expert while ensuring rational and predictable behavior after changes in the configuration.

If the new configuration is adopted, the best solution *before* making the configuration change and the best solution *after* making the configuration change become an new reference ranking pair added to the new database associated to the new configuration. In the pair, the best solution *after* making the configuration change is ranked first, and the best solution *before* making the configuration change is ranked second. All data influencing the overall decision function, such as the constraints and the additional factors introduced in Section 2.7 must be stored together with the pair to be able to apply the resulting new decision function in the old context, given the new configuration. Table 5 lists the consistency test in procedural form.

Human expert can specify implicitly the overall configuration of the constraints by asserting a set of 'normal' reference rankings. The easiest way to apply the heuristic that establishes consistent configuration parameters for the constraints is to let the human expert do parameter changes, and to later check them out with the introduced consistency test.

Another possibility would be that a machine learning scheme in combination with a 'teacher', being either a human expert or some objective a-posteriori meta-evaluation, could automatically establish variations to the configuration parameters. This combination should find the ranking intended by the teacher using the new configuration, i.e. solve the 'inverse' problem. The difficulty is, however, that humans easily overlook some constraints, especially when the number of constraints is large and the constraints are only vaguely defined. Therefore, the subjective 'better' ranking obtained a-priori from a human expert will often objectively not be better than the instantiation found by the system because the human expert forgot some constraints, thus forcing the system to learn suboptimal decision making. Therefore, the fine tuning scenario, where human experts repeatedly change constraint parameters such as weights by hand and then compare the respective best solutions, is much better suited to establishing the best configuration for the problem. It is however an open research problem whether this fine tuning can be fully automated when a non-human error-prone but objective, a-posteriori meta-evaluation is used, such as one guided by results of quality evaluations.

2.7 Decision function and conflict identification

To evaluate a given instantiation of a soft constraint satisfaction problem, a decision function aggregating all the constraints with their respective priorities, using an appropriate aggregation operator and a corresponding weighting scheme, must be established. The previous sections described possible techniques that help to construct such a decision function.

When scheduling, it is often advisable to introduce an additional measure into the decision function dependent upon whether the current schedule (= instantiation of the constraint satisfaction problem) contains certain difficult jobs. If the scheduling of these jobs is not introduced as a bonus into the decision function, these jobs might never be considered for actual scheduling. There usually exists a non-empty pool of waiting jobs, and only a subset of jobs from the pool can be scheduled immediately. Therefore, the danger is that some difficult jobs will remain in the pool forever unless additional measures are taken. It is clear that this 'difficulty' or 'importance' of a job must increase over the time for which it is still reasonable to 'produce' it, to favorize

```
procedure consistency_test(
  current_pool,
  old_configuration,
  new_configuration,
  old_reference_ranking_pair_database) {
  new_reference_ranking_pair_database ←
    new reference_ranking_pair_database;
  new_reference_ranking_pair_database.add(
    reference_ranking_pair(
      search_solution(current_pool, old_configuration),
      search_solution(current_pool, new_configuration)));
  consistency_flag ← true;
  for each reference_ranking_pair(
             best_solution_before_change,
             best_solution_after_change)
  in old_reference_ranking_pair_database {
    if satisfaction(best_solution_before_change, new_configuration) ≤
      satisfaction(best_solution_after_change, new_configuration)
    then {                                              // old ordering observed
      new_reference_ranking_pair_database.add(
        reference_ranking_pair(
          best_solution_before_change,
          best_solution_after_change));
    }
    else {                                              // old ordering violated
      outstream << "Inconsistent ranking "
                << reference_ranking_pair(
                     best_solution_before_change,
                     best_solution_after_change)
                << " (is not added to the new database)."
                << flush;
      consistency_flag ← false;
    }
  }
  new_configuration.save(
    new_reference_ranking_pair_database.save(consistency_flag));
  return consistency_flag;
}
```

Table 5: Pseudo-code of a consistency test for configuration changes. The program searches a database of reference ranking pairs to determine and report all those pairs in conflict the new parameters. It also builds up a new database in case the human expert wants to adopt the change, even if it is inconsistent with some old reference ranking pairs. Thus a hierarchical tree of variants and extensions of configurations, together with the corresponding database, is built up and named interactively by the human expert. If the consistency test returned true, the new configuration is an extension, otherwise it is a variant of the old one.

its eventual scheduling. The easiest way to introduce this 'difficulty' is to formulate a corresponding constraint with an associated priority that will represent these difficult jobs. Thus the 'difficulty' of jobs will be one criteria considered when the soft constraint satisfaction problem is optimized. The same applies equally to other soft constraint satisfaction problems such as those encountered in design or planning.

To guide the search in repair based algorithms as discussed in Section 3.1, it is necessary to identify the constraint with the worst weighted evaluation, i.e. the severest conflict which can be attacked to minimize conflicts. This can be considered as a side product of evaluating the current instantiation. It corresponds to computing the evaluation using the minimum operator, and more importantly, to remember the constraint involved in the minimal weighted evaluation. This constraint represents the largest conflict for the current instantiation. Often the constraint corresponds to a general feature of the instantiation and cannot be attributed

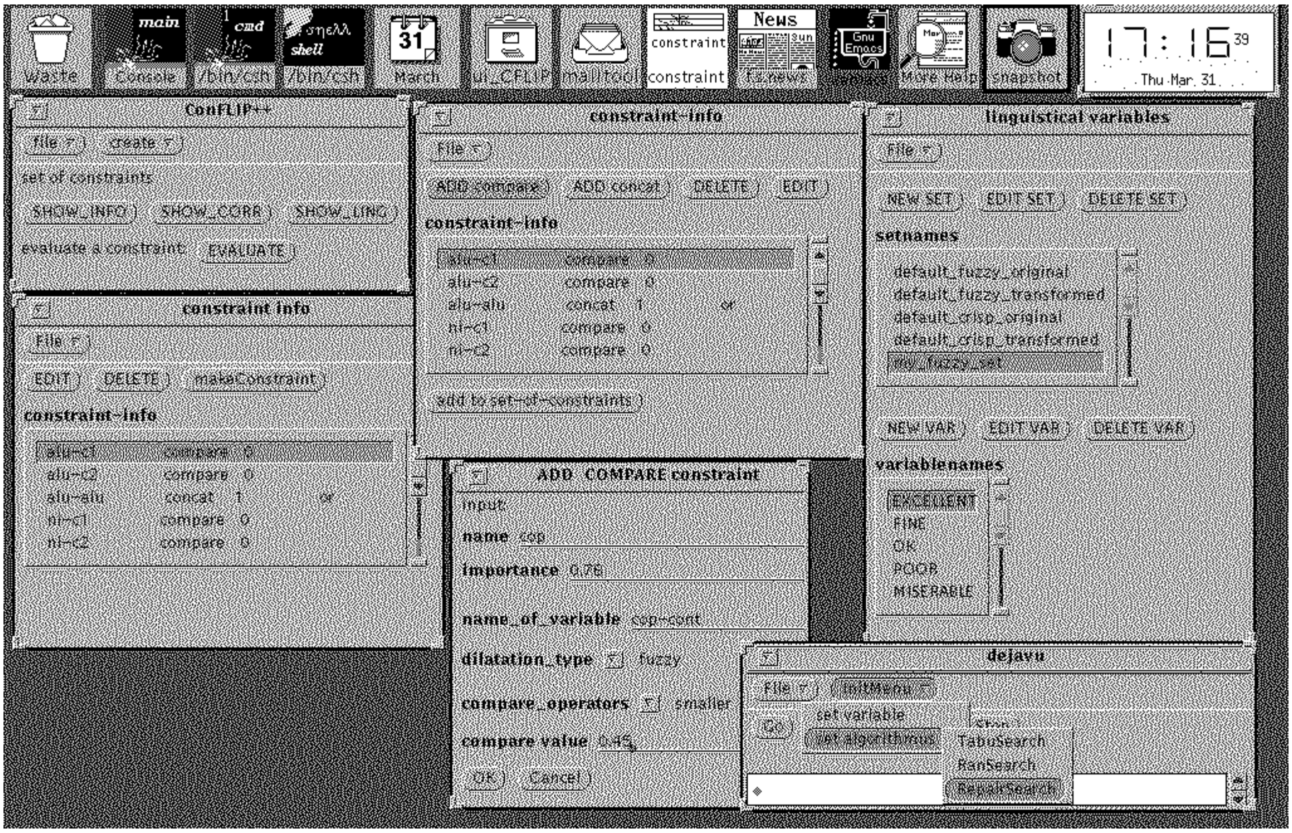

Figure 3: Interactive ConFLIP++ session in XView.

to a specific part of the instantiation. Depending on the repair operators available to the repair based constraint satisfaction algorithms, it can be helpful to find additionally the second largest and third largest conflict. Generally, the search should return the largest conflict being of a type that can be handled by an available repair operator.

2.8 Implementation issues and results with ConFLIP++

The reusable C++ object library ConFLIP++ is the result of the implementation efforts to realize the concepts introduced in the previous sections. It is a constraint-handling extension to FLIP++, which itself is a general purpose fuzzy logic inference process library. FLIP++ handles everything concerning fuzzification, membership functions, and linguistic variables. The user can choose between several different fuzzy inference methods, various priority schemes, different aggregation operators, and several defuzzification methods. FLIP++ also permits the graphical editing of membership functions and the easy manipulation of rule sets. ConFLIP++ itself is used as a knowledge representation tool in DéjàVu, the latter being a reusable scheduling library implementing various repair based search algorithms to compute schedules. See Section 3.2 for more information on DéjàVu. All three mentioned libraries have been developed at the Christian Doppler Laboratory for Expert Systems and are currently being further enhanced. Additionally, all these libraries have consistent user interface methods based on the public domain wxWindows toolkit from Julian Smart of the Artificial Intelligence Applications Institute at the University of Edinburgh. XView, Motif, or Microsoft Windows surfaces can be generated by just switching compile options. The FLIP++, ConFLIP++, and DéjàVu libraries are developed simultaneously on Sparc workstations and 386/486-PCs using Gnu-C++ as the programming language. Figure 3 shows a interactive ConFLIP++ session in XView.

ConFLIP++ allows to create, interactively edit, save and reload named sets of constraints including all parameters, and evaluate soft constraints. ConFLIP++ thus serves as a knowledge engineering tool in which domain knowledge can be stored, manipulated, and used for reasoning independently from the rest of the program using it. It is easy to add, modify, and maintain knowledge of several knowledge bases, while retaining maximal flexibility for constraints that can be adapted to model non-linear criteria used for optimization.

Using ConFLIP++, first simple constraints like

$$\text{aluminum-content} \quad \leq \quad 0.08 \tag{38}$$

are created and named, e.g. 'aluminum-constraint-38', using the objects and methods defined in ConFLIP++. The aim is to catch the vagueness in (38) where the $\leq$ sign is not meant to be interpreted in its strict mathematical sense, but such that 'smaller' violations are acceptable. What these 'smaller' violations could be has to be defined explicitly (and precisely) through the membership functions associated to the 'terms' of the variable as defined below. Additionally, ConFLIP++ is able to handle uncertainty about the exact value of 'aluminum-content', which is possible by propagating possibility distributions instead of defuzzified values. This capability to model *with accuracy* vague relations and uncertain data is the major contribution of fuzzy and possibilistic logics.

In (38), 'aluminum-content' is a so-called *linguistic variable*, a generalization of the conventional concept of a variable. Zadeh [25] defined a linguistic variable as a quintuple $\langle x, \mathcal{T}(x), U, G, \widetilde{M}\rangle$. x is the name of the linguistic variable, e.g. the character string "aluminum-content". $\mathcal{T}(x)$ is the term set of the linguistic variable, in this case a set of several strings such as {"negative_big", "negative_medium", "negative_small", "zero", "positive_small", "positive_medium", "positive_big"}. U is the universe of discourse, here for instance the range $[0, 100]$ since 'aluminum-content' is a percentage. G is the set of syntactic rules that generate terms. These rules are mainly important when terms can be further modified by general linguistic modifiers such as 'very'. $\widetilde{M}$ is the set of semantic rules that assign meanings to the terms. These rules define the membership functions relating the instantiation of a linguistic variable to the terms in the usual fuzzy set theoretic way. For example, Figure 1 defines the semantics of a term such as 'About_6'. In the case of 'About_6', writing down the underlying mathematical relationship through partially linear functions would be equivalent to specifying $\widetilde{M}$ for the corresponding linguistic variable $x_3 \stackrel{\text{def}}{=} x_1 + x_2$ through the corresponding graph in Figure 1.

Additionally, crisp constraints are predefined as a convenience for the human expert, but are internally mapped to specially tuned soft constraints.

In the next step, several such constraints are logically combined, i.e. they are aggregated by one of the operators introduced in Section 2.3, such as for instance 'aluminum-constraint-38 $\wedge$ nickel-constraint', to build-up more and more complex constraints. ConFLIP++ then automatically creates a ruleset out of default or user-defined term sets for standard linguistic variables, standard rule set tables, standard membership functions for the term sets, default priority values, and various default operators using FLIP++. FLIP++ is called again later to evaluate the constraint for some instantiations of the free linguistic variables appearing in the constraint. Additionally, the system checks the scores of all constraint having a priority different from zero as well as of their constituent subconstraints before these constraints are aggregated to find out whether a hard constraint violation occurred (evaluation score equals zero) in order to invalidate instantiations that crossed the hard barrier of the corresponding constraint.

The ruleset is built for instance such that, if the first linguistic variable is compared to its term 'positive_big', and the involved inequality is 'variable $\geq$ constant', and another linguistic variable is compared to 'zero', and the constraints corresponding to the two linguistic variables are concatenated by 'or', then the resulting term for the aggregated rule is 'very_good'. The latter term comes from the predefined standard term set {"very_good", "good", "zero", "bad", "very_bad"} and its associated $\widetilde{M}$ for output linguistic variables that normally will correspond to constraint satisfaction scores.

The human expert will usually have to fine tune the automatically created ruleset and the membership functions associated to the terms. However, it is possible to store user-defined standard sets of term sets and an associated $\widetilde{M}$. Additionally, fuzzy methods are quite robust, such that the exact determination of the membership functions is not essential. The predefined triangular membership functions often perform well in a first approximation. Nevertheless, one reason that makes fine tuning necessary is that ConFLIP++ has no a-priori domain knowledge: If the constraint is 'aluminum-content $\leq$ 0.08', some generated default rules are for instance:

$$\begin{array}{l} \vdots \\ \text{IF aluminum-content } \textit{is} \text{ positive\_small THEN aluminum-constraint-38 } \textit{is} \text{ zero} \\ \text{IF aluminum-content } \textit{is} \text{ positive\_medium THEN aluminum-constraint-38 } \textit{is} \text{ bad} \\ \text{IF aluminum-content } \textit{is} \text{ positive\_big THEN aluminum-constraint-38 } \textit{is} \text{ very\_bad} \end{array} \tag{39}$$

Here the semantics $\widetilde{M}$ will be recalculated by ConFLIP++ to assign membership functions to the terms fitting the central value 0.08. However, there is no information about the point starting from which a value is very large compared to 0.08 in this context. The standard 'support' (the domain where the membership function is larger than zero) of the term 'positive_big' might for instance be preset to [0.12,100] through the standard membership function calculation. Maybe 0.09 is meant to be already very high in this context, so that even a value that is classified 'positive_small' could be very bad and therefore the constraint would not be satisfied very well and should get a 'very_bad' evaluation instead. Therefore, the standard $\widetilde{M}$, defining the membership functions associated to the terms, will often not be sufficient, and sometimes even the terms' names will have to be edited in the rule set.

The next step is the evaluation of a constraint. The evaluation happens according to the ruleset of the constraint. First, the free linguistic variables have to be given values, the latter being either defuzzified reals or possibility distributions themselves. The evaluation function returns by default a defuzzified value that describes the degree of satisfaction of the constraint with the given values.

The human expert can influence the decision making behavior of ConFLIP++ in various ways. After a constraint knowledge base has been compiled, it can be copied and the copy can then be edited. First, the human expert can select one of several aggregation, implication, and defuzzification operators. The weighting scheme can be chosen as well. Of course, the individual membership functions and priorities of the constraints can be graphically edited. For instance, it is easy to selectively edit the constraint responsible for the observation of due-dates. These changes will have immediate effect on the decision function. To ease configuration of a complete constraint knowledge base built up from scratch, the default values for all these parameters are prespecified in a way that seems to apply reasonably well to most cases. However, the human expert can later soften or harden all those constraints that have not yet been fine tuned on a individual base. In such a case, ConFLIP++ searches the complete knowledge base for membership functions of linguistic variables that have not yet been edited by the human expert. These membership functions are replaced with new standard membership functions that make the decision making behavior of the constraint knowledge base fuzzier or crisper. Additionally, the human expert can load different configurations that were constructed earlier, as described in Section 2.6. These configurations can be saved as files from ConFLIP++ together with the database for reference rankings of pairs of instantiations needed for the consistency test described in Section 2.6. The test itself is available as a method of the ConFLIP++ library, and gets automatically invoked whenever changes in the configuration are made.

3 Fuzzy multiple criteria optimization

The methods introduced in Section 2 assume implicitly that *all* possible evaluations of instantiations can be compared among one another to find the best solution. However, in real-world problems, this search space may well be very large or even infinite when variables have continuous domains. The problem of efficiently evaluating all the alternative solutions becomes quickly intractable in such a case. The problem is not solvable in its totality in any reasonable amount of time unless some type of heuristic is applied. However, there is generally no need to really find the optimal solution for real-world problems. Instead, settling for a slightly suboptimal but good solution found using a heuristic is often acceptable, supposing that it was found by using all the intelligence, computing power, and time available for that particular purpose.

In the following sections, the techniques introduced in Section 2 are applied to the scheduling of a steelmaking plant: Section 3.1 explains the general repair based strategy that has been successfully applied to problems which until recently were believed to be completely out of the reach of present day computers. Section 3.2 goes on to detail the combination of such repair based techniques with the methods introduced in Section 2, culminating in the DéjàVu implementation for fine-grain scheduling of a steelmaking plant.

3.1 Fuzzy repair

General aspects of fuzzy constraint satisfaction problems have been discussed in Section 2.1. Most of the approaches to fuzzy constraints in the literature have used methods from operations research [27] or standard backtracking algorithms [3, 7, 8, 12]. In the backtracking case, the decision tree that must be searched grows exponentially with the problem size. It is therefore necessary to prune the search tree using heuristics. Guan [12] for instance uses a coefficient λ_0 to truncate all paths for which the satisfaction falls below λ_0, corresponding to shifting the hard barrier to more constraining values. Similar methods must be used by all constructive backtracking methods. The problem is however that no compensation with partial solutions evaluating below λ_0 can be envisaged.

The problems solved by applying methods from operations research are mainly limited to linear problems or must use heuristics too, in which case the remarks made for the standard backtracking algorithms apply as

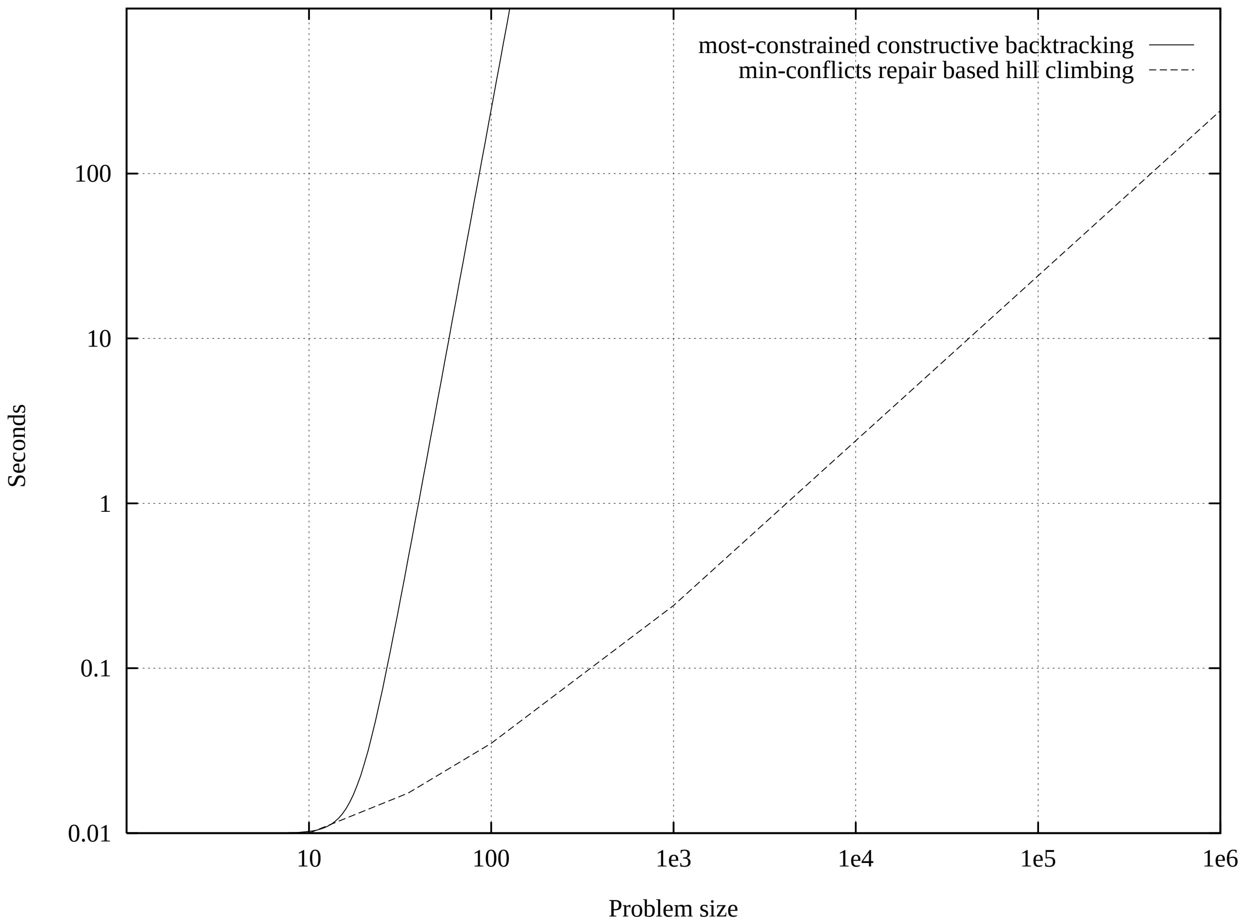

Figure 4: Mean solution time for min-conflicts repair based hill climbing method on n-queens problem averaged over 100 runs for several different n versus the most-constrained constructive backtracking method, the best general constructive method known (Results taken from Minton et al. [16]). Note that the repair based method exhibits linear time and space complexity for large n. The million queens problem was consistently solved in less than four minutes on a Sparc workstation 1 by the repair algorithm.

well.

In contrast, repair based heuristics (sometimes called 'iterative improvement techniques') have successfully been applied to solve problems that previously seemed intractable. They have generally scored much better than constructive approaches that start from an empty instantiation and build successively more and more complete partial instantiations, such as constructive backtracking methods. Additionally, non-linearities in the constraints pose no difficulty for repair based approaches since they can easily work with approximate models of these constraints while yielding correct results. Minton et al. [16] have analysed the solution to the problem of scheduling the Hubble space telescope, a complex task by almost any measure. They have investigated the underlying heuristics, testing it on cases such as the n-queens problem and the graph-coloring problem. These problems have served for many years as classical benchmarks to study the efficiency of new heuristics and algorithms that solve constraint satisfaction problems.

The results were very promising, as the general repair based algorithm proposed by Minton et al. [16] could find solutions in less than four minutes on a Sparc workstation 1 for the *million*-queens problem, while the best constructive approach (found in an empirical study by Stone and Stone [21] to be the most-constrained backtracking algorithm) became intractable for $n > 1000$. Figure 4 summarizes the collected statistics averaged over 100 runs for several different n. Minton et al. even found that their repair based method exhibits linear time and space complexity for large n. The min-conflicts heuristic combined with a repair based hill climbing heuristic specifies that, starting from an initial suboptimal solution, the system attempts to minimize the number of constraint violations after each repair step. They showed convincingly that for certain problems, the use of the additional knowledge gained from operating on complete but suboptimal solutions instead of building solutions from scratch as in constructive approaches pays off well. Such repair based heuristics perform orders of magnitude better than traditional backtracking techniques. Though repair based methods can be combined with many general search strategies, they found that hill climbing methods were especially well suited for the problems they investigated. Guan [12] benchmarked constructive backtracking algorithms on the n-queens problem modeled with fuzzy constraints, the results being in line with those found by Minton et al. [16].

In general, scheduling problems appear to be excellent applications for repair based methods, because:

- The general algorithm is simple and therefore easy to understand and to implement.
- Repair based methods can be naturally adapted for rescheduling [28], the latter being often necessary in real-world situations because of Murphy's law, i.e. machines break down. Additionally, it normally takes much less time to change a solution than to build a new solution from scratch.
- When rescheduling occurs, the *initial* solution is normally taken to start the search for a new solution observing the additional constraints introduced by the new situation. The new solution will therefore automatically be similar to the old solution, ensuring that as few changes as possible have to be implemented. Because of this feature, the resulting overall scheduling system can be termed *robust* concerning changes. Biefeld and Cooper [2] report in this connection that human schedulers found repair based methods very natural and similar to their way of thinking.
- Repair based methods inherently posses the 'anytime' property: They return a good solution from the beginning, but can continue the search for a very good solution as long as there is time left.
- Repair based scheduling methods can easily be used interactively, sometimes termed 'cooperative scheduling', since the human expert can propose an instantiation that can be used as a starting point from which the iterative repair method resolves as many conflicts as possible.
- By combining repair based methods with concepts such as fuzzy constraints, it is easy and natural to optimize the scheduling solution while repairing the current suboptimal schedule.

Minton et al. [16] note that one limitation of the repair based hill climbing heuristic is that, as any greedy search method, it can become stuck in pathological cycles. One solution to this problem is the combination of the repair based hill climbing heuristic with the tabu list technique that was included in the DéjàVu scheduling library described in Section 3.2. The results obtained with this technique were consistently better than by using the simple repair based hill climbing heuristic alone.

More generally, Selman et al. [18] have shown that a repair based algorithm performs well on hard satisfiability problems. This indicates that the underlying technique is more useful to solve difficult problems heuristically than is generally assumed.

3.2 An application example: Scheduling a steelmaking plant with DéjàVu

A detailed description of the steelmaking scheduling problem and its solution can be found in Dorn and Slany [4], the complete description of the employed search algorithms and their performance in Dorn et al. [5]. The main characteristics are shortly restated here to enable the reader to understand the scheduling context. Steel is produced on two continuous caster units, one being double stranded. Each day a cast sequence of about 35 charges (jobs) is scheduled for one caster over the next 24 hours. Hot pig iron is delivered from the blast furnace to the LD-converters. The steel is then poured into ladles, processed further in secondary metallurgy aggregates, and finally delivered to the casters. The casters produce continuously strands of steel that are cut into slabs of specific length. Although the steel is produced in different aggregates in a fixed flow, the casters are the bottleneck resource of the shop. The tundish, a part of the caster, has to be maintained after approximately 240 minutes. A detailed analysis of this variable's on-line determination is given in Slany [19]. The maintenance takes approximately 100 minutes, therefore a second tundish is used while the first is maintained.

The main criteria that constrain the sequence of charges are compatibility constraints between jobs. Three compatibility aspects must be considered in order to optimize production quality:

- The steel grade (the chemical analysis) of subsequent charges must be similar.
- The casting format may vary only within certain limits.
- The degassing procedure in the secondary metallurgy aggregates must be compatible between subsequent charges.

The rules are explicitly given in a crisp form, but human operators have been observed to 'break' or relax these a-priori hard rules and to implicitly make trade-offs in 20% of their decisions. Their behavior can be adequately modeled by using soft constraints with hard barriers as developed in Section 2.

To optimize the production, the casters should process the steel without interruption. Various operations can help when hard barriers otherwise would be exceeded. These include inserting a steel plate into the strand to separate the different qualities such that they do not mix too much, or to prematurely exchange the tundish in use with the other one. However, both operations have associated costs. If the tundish-exchange caused by quality considerations can be combined with a tundish-exchange because of tundish maintenance considerations, these costs can be minimized. The most expensive operation is to stop and then setup from scratch the whole casting unit. This takes a lot of time, but is sometimes necessary to do maintenance on the caster, or to produce certain difficult jobs.

There are soft temporal release-date and due-date constraints for certain jobs that should be delivered still hot to a subsequent plant. However, since there are (limited) warm-holding places available, the exact observance of temporal constraints is not obligatory. Additionally, the temporal constraints can be renegotiated with subsequent plants, and sometimes it is possible to exchange the temporal constraints of two jobs.

Generally speaking, it is possible to find a feasible schedule. However, criteria such as produced quality, tundish usage time, quality separating operations, number of setups and maintenance intervals of machines, release-dates, due-dates, and even such aspects as work-load of staff should all be optimized. Additionally, rescheduling has to be done quite frequently when some production parameters change due to machine break-downs. In the past, most human errors were made in these rescheduling situations since time to think is scarce and the situation worsens rapidly (e.g. forgetting for some time a waiting aggregate, resulting in longer waiting times or worse qualities for certain jobs) if no action is taken.

In a first expert system approach, Stohl et al. [20] applied a constructive domain heuristic to the problem. Although the system found good feasible solutions, Stohl et al. believe that the solutions could be further improved, especially since constraints could only be broken through explicit user-intervention, and because the relaxing of constraints was not evaluated.

DéjàVu is a reusable library for scheduling problems implemented in C++. The name 'DéjàVu' (French for 'seen before') indicates that the methods of the library can be reused and therefore will usually be encountered more than once. So far, it has been applied to the daily fine-grain scheduling of the steelmaking plant LD3 of the VOEST Alpine Stahl AG in Linz, Austria. There are concrete plans to reuse as much as possible of the library when applying it to other problems in the future.

Several repair based algorithms were integrated in DéjàVu, 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.

All algorithms are repair based and have several variants and different parameters. The conflict identification function explained in Section 2.7 is used together with a domain dependent repair operator library to quickly choose the repair operator that will most probably minimize conflicts for a given situation. However, the algorithms are independent of this library since the guidance provided through the conflict identification function is in all cases combined with a fallback random strategy if nothing else helps to find better instantiations [5]. Figure 5 shows some results for the tabu list min-conflicts repair based hill climbing heuristic versus the min-conflicts repair based random search hill climbing heuristic, for 10 samples each, with a fixed bad random initialization evaluating to 0.58139 for the chosen configuration, versus the constructive backtracking heuristic. The repair based timings were taken on a 386-PC and not on a Sparc workstation as for the constructive approach. Results using the other repair based algorithms (iterative deepening, genetic algorithm) implemented in DéjàVu so far yield intermediate results, almost equal to those found by the tabu list hill climbing heuristic. As one sees in Figure 5, simple random search repair is easier trapped in suboptimal solutions. Starting from a better initial instantiation (for example from the solution found with the constructive approach) does not really influence results taken after three minutes run-time. More statistics and a detailed analysis of the different algorithms can be found in Dorn et al. [5]. All repair based heuristics are much faster and yield better results than the constructive approach that was evaluated using the same configuration parameters. However, the timing results are not really comparable because of different programming/run-time environments and slightly

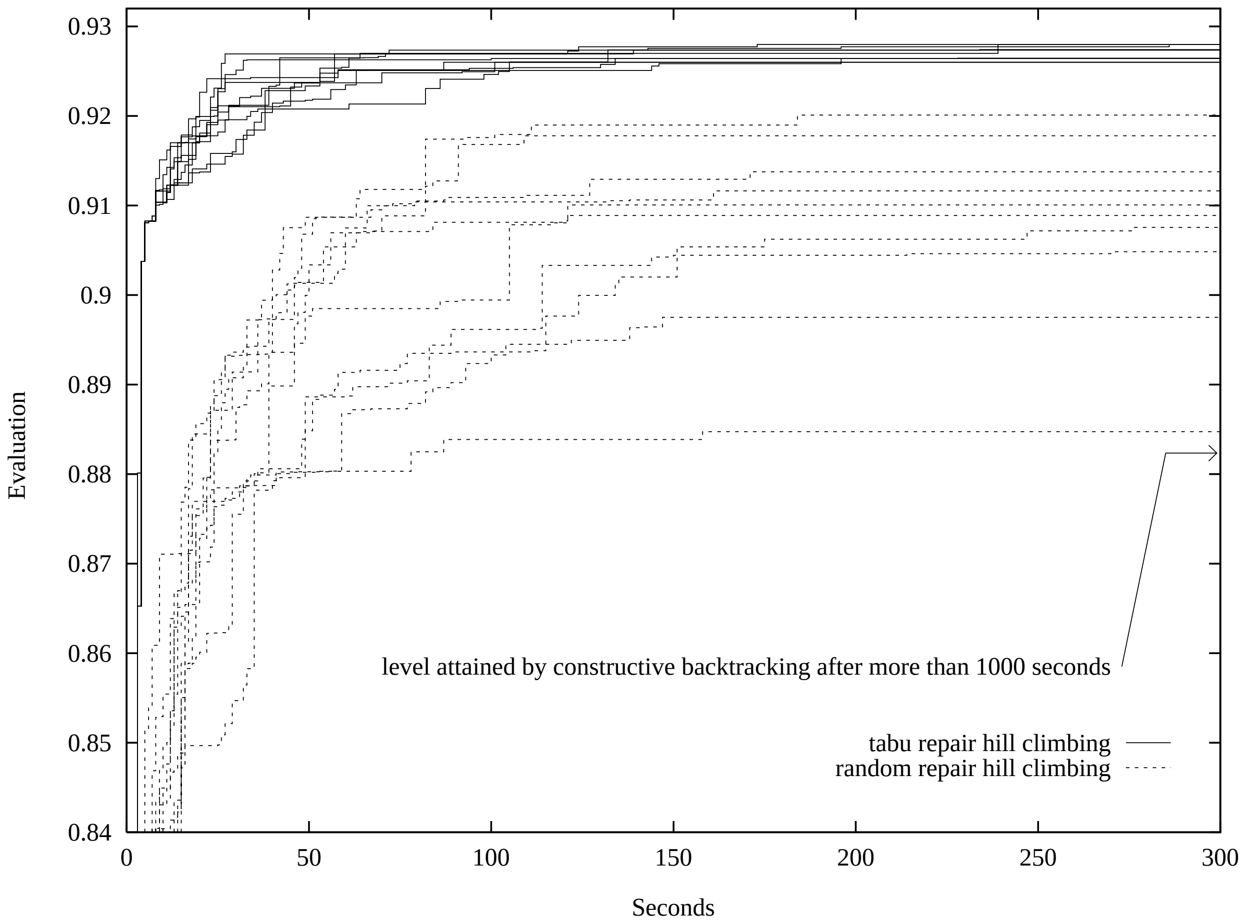

Figure 5: Tabu list min-conflicts repair based hill climbing heuristic versus min-conflicts repair based random search hill climbing heuristic, 10 samples each, with a fixed bad random initialization, versus constructive backtracking heuristic.

different rule sets used to search solutions in the constructive backtracking case, thus leading to slightly different optimal solutions.

4 Conclusion

We developed a combination of repair based methods and fuzzy constraints for real-world multi criteria decision making, with a bias towards scheduling problems. We presented improved methods for compromising between antagonistic criteria, for assessing priorities among fuzzy constraints, as well as a new method for ensuring consistent and reasonable changes in configurations.

The results obtained from a steelmaking application indicate the superiority of our approach compared to constructive non-fuzzy methods in terms of modeling expressiveness and performance.

Acknowledgements

I would like to thank Markus Bonner, Gerhilde Egghart, Mario Girsch, Thomas Länger, Stefan Mayer, and Günther Skele for their excellent implementation work and many valuable discussions. I especially wish to thank Jürgen Dorn, Georg Gottlob, Peter Klement, and Norbert Schindler for their assistance and advice.

References

[1] R. E. Bellman and L. A. Zadeh. Decision-making in a fuzzy environment. *Management Science*, 17(4):141–164, December 1970.

[2] E. Biefeld and L. Cooper. Bottleneck identification using process chronologies. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'91)*, Sidney, Australia. 1991.

[3] Dahai Chen, Yuhuan Pan, and Jinsong Xue. A fuzzy production system with backtracking control strategy for multiobjective scheduling to a one-machine-n-parts problem. In Andrew Kusiak, editor, *Modern Production Management Systems*, pages 135–145. Elsevier, 1987.

[4] Jürgen Dorn and Wolfgang Slany. A flow shop with compatibility constraints in a steelmaking plant. In Mark Fox and Monte Zweben, editors, *Intelligent Scheduling*. Morgan Kaufmann, 1994.

[5] Jürgen Dorn, Mario Girsch, Günther Skele, and Wolfgang Slany. Comparison of iterative improvement techniques for schedule optimization. CD-TR 94/61, Christian Doppler Laboratory for Expert Systems, Technical University of Vienna, 1994.

[6] Didier Dubois and Henri Prade. *Possibility Theory: An Approach to Computerized Processing of Uncertainty*. Plenum Press, N.Y., 1988.

[7] Didier Dubois, Hélène Fargier, and Henri Prade. Handling flexibility and uncertainty in job-shop scheduling. Rapport IRIT 93-30-R, Institut de Recherche en Informatique de Toulouse (IRIT), November 1993.

[8] Hélène Fargier, Jérôme Lang, and Thomas Schiex. Selecting preferred solutions in fuzzy constraint satisfaction problems. In Hans-Jürgen Zimmermann, editor, *EUFIT'93, First European Congress on Fuzzy and Intelligent Technologies*, pages 1128–1134, Aachen, Germany. Augustinus Buchhandlung, September 1993.

[9] Eugene C. Freuder and Richard J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58:21–70, 1992.

[10] Fred Glover. Tabu search–part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[11] Fred Glover. Tabu search–part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[12] Qi Guan. *Extending Constraint Satisfaction Problem Solving with Fuzzy Set Theory*. PhD thesis, Technical University of Vienna, 1994.

[13] Itsuo Hatono, Tetsuya Suzuka, Motohide Umano, and Hiroyuki Tamura. Towards intelligent scheduling for flexible manufacturing: Application of fuzzy inference to realize high-variety of objectives. In Ming Leu, editor, *Proceedings of the JAPAN/USA Symposium on Flexible Automation (A Pacific Rim Conference)*, volume 1, pages 433–440, San Francisco, California. July 1992.

[14] Ahmed Ibrahim and Bilal M. Ayyub. Multi-criteria ranking of components according to their priority for inspection. *Fuzzy Sets and Systems*, 48(1):1–14, May 1992.

[15] Roger Martin-Clouaire. Dealing with soft constraints in a constraint satisfaction problem. In *Proceedings of the 4th International Conference on Information Processing of Uncertainty in Knowledge-Based Systems*, pages 37–40, Palma de Mallorca, Spain. July 1992.

[16] Steven Minton, Mark D. Johnston, Andrew B. Philips, and Philip Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.

[17] T. L. Saaty. *The Analytic Hierarchy Process*. McGraw-Hill, 1980.

[18] B. Selman, H. J. Levesque, and D. Mitchell. A new method for solving hard satisfyability problems. In *Proceedings of the AAAI'92 Conference*, San Jose, Ca. 1992.

[19] Wolfgang Slany. Fuzzy expert system to predict maintenance intervals in a continuous caster. In Duk-Hyon Baik, editor, *Preprints of the international conference CPC-93, Computerized Production Control in Steel Plants*, pages 291–296, Seoul, Korea. The Korean Institute of Metals and Materials, The Institute of Materials, UK., November 1993.

[20] K. Stohl, W. Snopek, T. Weigert, and T. Moritz. Development of a scheduling expert system for a steelplant. In *Proceedings of the IFIP Conference on Experts Systems in Mineral and Metallurgy*, Finland. 1991.

[21] H. S. Stone and J. M. Stone. Efficient search techniques – an empirical study of the n-queens problem. *IBM Journal of Research and Development*, 31:464–474, 1987.

[22] Ronald R. Yager. A new methodology for ordinal multiobjective decisions based on fuzzy sets. *Decision Sciences*, 12:589–600, 1981.

[23] Ronald R. Yager. On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Transactions on Systems, Man, Cybernetics*, 18:183–190, 1988.

[24] Lotfi A. Zadeh. Fuzzy sets. *Information and Control, New York: Academic Press.*, 8:338–353, 1965.

[25] Lotfi A. Zadeh. The concept of a linguistic variable and its application to approximate reasoning (parts 1 and 2). *Information Sciences*, 8:199–249, 301–357, 1975.

[26] Hans-Jürgen Zimmermann. *Fuzzy Set Theory — and Its Applications.* Kluwer Academic Publishers, 2nd, revised edition, 1991.

[27] Hans-Jürgen Zimmermann. Fuzzy mathematical programming. In Stuart C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, volume 1, pages 521–528. John Wiley & Sons, Inc., 2nd enlarged and revised edition, 1992.

[28] Monte Zweben, M. Deale, and R. Gargan. Anytime rescheduling. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning and Scheduling*, 1990.