Representation and Processing of Preferential Rules

Ulrich Geske[1], Hans-Joachim Goltz[2], Armin Wolf[2]

[1]University of Potsdam, [2]Fraunhofer FIRST, Berlin
Ulrich.Geske@uni-potsdam.de
{Hans-Joachim.Goltz, Armin.Wolf}@first.fraunhofer.de

Abstract: Preferential rules occur very often in application domains like timetabling. Besides absolutely valid rules like "Ground courses have to start in the morning in one of the main lecture rooms", there are very often preferential rules like "A wish for the starting time of a lecture should be satisfied with less deviations" or "A wish for a room of a lecture should be satisfied if possible". Our aim is to develop easy-to-use techniques for applying preferential rules and to demonstrate their use and efficiency. We are motivated by the non-availability of suitable implementations and the need of efficient execution.

1 Introduction

The problem, which results from the two types of knowledge (Fig.1) – absolutely valid rules and preferential rules – consists in combining the calculi to obtain a consistent and efficient procedure for knowledge processing. A useful paradigm for dealing with combinatorial complex problems, such as timetabling, is their formulation as Constraint Satisfaction Problems (CSP). A CSP is defined by P = (V, D, C) with a set of variables $V = \{V_1,...,V_n\}$, a domain of values for each of the variables $D = \{D_1,...,D_n\}$, and a set of relations (constraints) $C = \{C_1,...C_m\}$. All C_i are considered as hard constraints (part of strict rules), i.e. they are true or false. A solution of a CSP is a consistent complete assignment of values from D to V, i.e. all constraints have to be satisfied. Besides the rules that must always be fulfilled there may be also rules in a problem that should be fulfilled if possible or in the best possible way. Such rules are called "wishes" and they can have different priorities. Some examples for wishes in timetabling are:

1. The wish of the lecturer for starting times with different priorities.
2. The wish for a special room for a teaching event (e.g., the main lecture hall).

Similar kind of wishes occur in other problems, like personnel planning (e.g. "Thursday in the fourth week should be free, if possible"}), maintenance planning ("Worker A should preferential serve device B, maybe also C"}), train scheduling ("A train may use platform A or B, but preferable A, in a station"), design of devices ("A part maybe composed of different groups of subparts"). The structure of the mentioned wishes partly sometimes differs. For instance, wish No.2 from the timetabling wishes given above can either be satisfied or not. Let us assume that there is no second main lecture hall nor any

other comparable room. Then, the last alternative is to assign an arbitrary room suitable for a large number of students. Let us call rules that deal with such wishes "if-possible rules". Another example of an if-possible rule is if a nurse wants to have a specific day off. Either there are enough staff available on that day such that the wish can be fulfilled or she has to work on that day. The wish cannot be fulfilled by giving her the day off a day later. Another situation invokes wish No.1 from the timetabling wishes listed above. The fulfilment of wishes begins with the wish that has the highest priority. If this wish cannot be fulfilled, the wish with the next-lower priority is considered. A similar structure occurs in the train scheduling example. Platform B should be used only if platform A is out of order or occupied by a train with higher priority. Let us call rules that realize the processing of wishes according to their priority "soft rules" (cf. Fig.1).

2 Modelling Preferential Rules

We are especially interested in an efficient approach which avoids the use of a meta-interpreter to deal with preferential rules. Using the built-in features of a constraint system, it is rather simple to model if-possible rules as Weighted CSP. For modelling soft rules we use either the concept of domain reduction [Go95] during search or the concept of constraint hierarchies. In the later case we transform the hierarchy for efficiency reasons into a "flat" system of "hard" constraints which can be processed by an ordinary system without the need for a meta-interpreter [Wo98], [Sch98], [FS04].

2.1 If-Possible Rules

The general idea behind the framework of Weighted CSP is to minimize weights (costs, penalties) for unsatisfied constraints. The most studied instance of weighted constraints, the MAX-CSP, is specialized such that the maximum number of constraints are satisfied. A weighted constraint is defined by

- a pair (c, w(c)) with c as a classical constraint and w(c) giving the weight of constraint c as an element of totally ordered set W,
- an ordering relation $\leq_w$ such that smaller weights represent weights of less important constraints,

A solution of an weighted CSP is such an assignments θ, that its satisfaction degree ω(θ) is minimal w.r.t. $\leq_w$. The satisfaction degree ω of assignment $\theta \in \Theta_V = D_1 x \ldots x D_n$ is given by the sum of weights of all constraints which are unsatisfied by the assignment θ. Fig.2 shows an example for a weighted CSP. There are three lectures. The unary constraints on each lecture mean that the *first* lecture should be the first one in a timetable, the *second* lecture the second one, and the *third* the third one. Each of these constraints has the weight 2, i.e. if such a constraint is not satisfied, it counts as 2 for the satisfaction degree. The binary constraint on *first* and *second* is a hard constraint and therefore has the highest possible weight (if it is unsatisfied, the satisfaction degree will be the worst). The weight of the binary constraint on *first* and *third* counts more than the unary constraints for the satisfaction degree but less than the weight of the other binary constraint. Thus the order *first, third, second* has the minimal satisfaction degree and

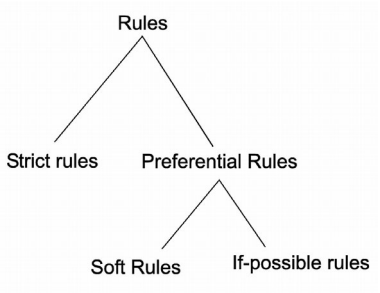

Fig. 1: Different kind of rules

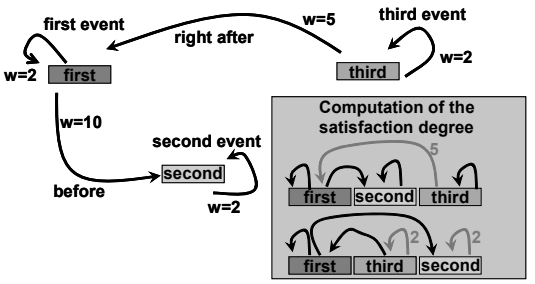

Fig. 2: Weighted CSP (example from [Ru01])

consequently it represents the solution of the scheduling problem. The essential kernel of a weighted CSP is the assignment of a weight to each constraint. The more important a constraint the higher is the weight. All weights of conjunctive constraints which cannot be satisfied are summarized. Such an assignment of values to variables which minimizes the sum of weights is the solution of the problem. In a first step, we seek to simplify the paradigm and allow only the weights 1 and ∞ (for strict rules), i.e. we count the number of non-satisfied constraints instead of their weights. We can transform this task into a logical specification using Boolean variables and the global constraint *among(Integer, Variables, Values),* which counts in the first argument *Integer* how many *Variables* take a value from the list of *Values*. The *among* constraint holds, if $\forall i \in [1,m-1]$: $V_i<V_{i+1}$ exactly N variables *among* $X_1,\ldots,X_k$ take their value in the list of values $V_1,\ldots,V_m$. In the special case *among(Bool,*$[X_i]$*,*$[V_1,\ldots,V_m]$*)* the variable X_i takes or does not take a value from the list of values $[V_1,\ldots,V_n]$. In the following timetabling example, the variables in *List* should denote the corresponding starting times for the lectures. There are in general 42 different possible starting times for lectures in a week of six days, 7 per day. In CHIP5 [Cos03] the *among* constraint could be used in the following way (*among/3* is an abstraction of the among/5 constraint of CHIP5):

```
if_possible_test(VarNo, List_of_Wishes, No_of_not_fulfilled_wishes):-
   length(List, VarN), List::0..42,
   if_possible_vars(List,List_of_Wishes, If_possible_List),
   satisfaction_degree(If_possible_List, No_of_not_fulfilled_wishes),
  min_max(labeling(List,0,first_fail,indomain), No_of_not_fulfilled_wishes).
```

The procedure *if_possible_vars/3* deals with each of the N variables by calling *if_possible/3*. Let us assume that for each of these variables Var_i (i.e. starting times of lectures) a list $List\_of\_Wishes_i$ of preferred values (wishes) is given. The among-constraints in *if_possible/3* cause $If\_possible_i$ to take the value 1 if Var_i reaches a value from $List\_of\_Wishes_i$ or 0 otherwise.

In order to check the fulfilment of all wishes, *satisfaction_degree/3* computes the *No_of_not_fulfilled_wishes* (in the following N is the number of variables Var_i). It expresses for how many of the variables no fulfilment of wishes occurs. Minimization of not fulfilled wishes takes for a relevant problem size (500 variables and wishes) about 1.1 seconds (Core2 Duo, 2GHz CPU, List_of_Wishes=[42]).

```
if_possible(Var,List_of_Wishes,If_possible) :-
  If_possible::0..1, among(If_possible,[Var],List_of_Wishes).

satisfaction_degree(If_possible_List, No_of_not_fulfilled_wishes):-
  length(If_possible_List, N) , No_of_not_fulfilled_wishes::0..N,
  among(No_of_not_fulfilled_wishes, If_possible_List,[0])
```

In the former discussion it is assumed that constraints have the same weight. This is a reasonable assumption for many problems, where only the number of not fulfilled wishes is important. Sometimes, as assumed in the Weighted CSP paradigm, it is important to include different weights. Let $p_1,\ldots,p_N$ be arbitrary weights of wishes. The maximum possible error can be computed by: ErrorMax = $\sum_{i=1}^{N} p_i$. The actual error of a solution is a value of the interval: Error :: 0..ErrorMax. To compute the satisfaction degree of a solution, the value of *If_possible*$_i$ of the i-the variable (0 = wish satisfied, 1 = wish not satisfied) should be considered: Error = ErrorMax - $\sum_{i=1}^{N}(p_i * \text{Bool}_i)$.

2.2 Modelling of Soft Rules

Unlike if-possible rules, there are alternatives to fulfil soft rules. These alternatives can be specified implicit or explicitly. A wish for a lecture to begin Tuesday at 11 a.m. could be explicitly given if Monday 11 a.m. is not possible. On the other hand, it is reasonable to extend the wish to begin on Monday at 11 a.m. by an interval of ±15 minutes or perhaps ±2 hours. This kind of weakening of demands facilitates the search for an acceptable solution.

2.2.1 Domain reduction method

The assignment of sets (instead of single values) to variables in the domain-reducing strategy is the method most compatible with the heuristic of postponing decisions as long as possible thus avoiding inefficiency by frequent backtracking (trashing) [Go95]:

- The assignment of a value to the selected variable is replaced by a reduction of the domain of this variable.
- If backtracking occurs, the unused part of the domain is taken as the new domain for a repeated application of this method.

A solution is narrowed down by this reduction procedure. A conventional search algorithm can be used for the final value assignment. If a contradiction is detected during the search, it has to backtrack into the domain reducing procedure. The heuristics for determining the size of the reduced domain take into account wishes with respect to starting times for lectures. In the first step, it is attempted to reduce the domains of the variables such that the expressed wishes are included in the reduced domain. More specifically, if t_p is a wish, then the interval $[t_{p-2}, t_{p+2}]$ is considered (i.e., a difference of two quarter of an hour in each direction). This is the default interval, if no other interval is specified. The time intervals created by preferred time points are called preferred intervals. In the second step, values are selected for all domain variables. For this value selection, a labelling algorithm is used which attempts to select the preferred values first.

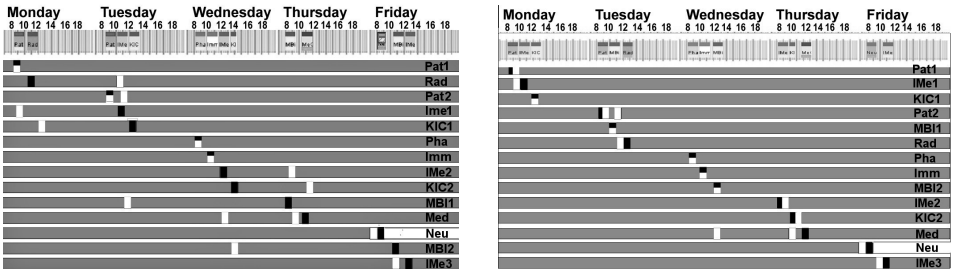

Figure 3 Timetabling: Wishes processed a) by strict rules, and b) by soft rules

A comparison of the timetabling procedure without and with domain reduction technique shows the advantage of the soft-constraint technique. In Fig. 3, a schedule of 14 lectures is presented in the headline. In the same order, the 14 grey lines represent the scheduled times (black; below lectures) and wishes (white). E.g., the second grey line in the Fig. 3a shows for the second lecture in the headline the starting time at Monday 10 o'clock and a wish at Tuesday 10 o'clock. Like here, there are often big differences between both times. Fig. 3b shows the effect of soft constraints (domain reduction). In most cases scheduled times and wishes have the same position (i.e. wishes are satisfied).

2.2.2 Constraint Hierarchies

Every constraint gets an additional argument specifying the hierarchy level or the weakness of the constraint. The weakness 0 denotes a required constraint that must be satisfied like a constraint (hard constraint) in the basic system (required constraints are transformed into hard constraints). The higher the number of this argument, the higher is the weakness of the constraint. In order to find the optimal values of the variables, an error function e_c is joined to the constraints. All errors of a level *i* are combined to give an error $E_i\sigma$ of this level for a certain instantiation σ of the variables. The goal is to find a instantiation σ so that no other instantiation θ supplies a better value for the total error $E(G\sigma)$ of the complete hierarchy. The following example shows a wish for the mathematics lecture. The level 0, the required constraints that have to be satisfied is empty. At the first soft level (1) a wish is called, and if it is not possible to satisfy it, there are alternatives at levels 2 and 3.

```
def_lecture  ( id('Mathematics for CS II'),
  possible_lecturer(['Lecturer56']),
  possible_room([room2,room3,room4,room22,room26,room27]),
  softconstraints( [c(0,[]),
                    c(1,[time_and_room([tue(15 : 15)],[room4])]),
                    c(2,[time_and_room([tue(13 : 15)],[room4])]),
                   c(3,[time_and_room([tue],          [room26])])],      [sum], _)).
```

To verify the efficiency of the soft-constraints approach, we have compared the time behaviour of different relations of hard/required and soft constraints for same problems. Runtime examinations (Fig. 4) prove that there is no loss of efficiency when using soft constraints. On the contrary, even for a large number of scheduled lectures, the use of

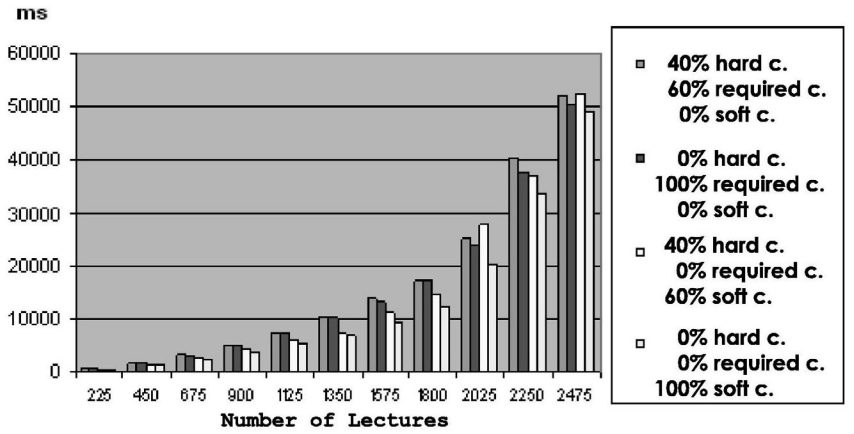

Figure 4 Comparison of efficiency [He05]

soft constraint is faster than the use of hard/required constraints. One explanation may be that there is more flexibility for the system to assign starting times to lectures and an avoidance of frequent backtracking steps.

3 Conclusion

Preferences frequently occur in practical problems. We have discussed the situation that there may be no available implementations of a known paradigm for dealing with preferential rules. We were able to demonstrate that constraint-based systems contain the basic facilities to model and process preferences in addition to strict rules. Our goal is not only the availability of easy-to-use modelling facilities but also to ensure efficiency of rule processing.

References

[Cos03] Cosytec SA, France : Finite Domain Constraint Reference Manual.Version 5, 2003.

[Go95] Goltz, H.-J.: Reducing domains for search in CLP(FD) and its application to job-shop scheduling. In Montanari, U.; Rossi, F. (eds.): Principles and Practice of Constraint Programming, CP'95}, LNCS (vol. 976), Springer-Verlag, pp 549-562, 1995.

[Hei05] Heimann, M. von: Constraint-based Optimization at the example of Timetabling. In German. Diploma Thesis. FHTW Berlin, 2005.

[FS04] Franz, Ch; Schmidt.T.: Soft-Constraint Library. In German. ITR-FhG 2004.

[Ru01] H. Rudova. Constraint Satisfaction with Preferences. Ph.D. Thesis. Masaryk University Brno. Faculty for Informatics. 2001

[Sch98] Schiemann, A.: Extension and Implementation of methods for solving constraint hierarchies using different error functions. Diploma Thesis. In German. Technical University of Berlin, 1998.

[Wo98] Wolf, A.: Solving hierarchies of finite-domain constraints. J. Expt. Theor. Artif. Intell. 10(1998)131-143.