Handling preferences in Constraint Logic Programming with Relational Optimization

François Fages[1,2], Julian Fowler[1] and Thierry Sola[1]

[1] Thomson-CSF/LCR, 91404 Orsay Cedex, France.
[2] LIENS CNRS,Ecole Normale Supérieure, 45 rue d'Ulm, 75005 Paris, France.

Abstract. In many Constraint Logic Programming (CLP) applications one needs to express not only strict requirements but also preferences. Constraint hierarchies are one way of describing preferred criteria in the statement of a problem. In [18] CLP was extended to integrate constraint hierarchies resulting in Hierarchical Constraint Logic Programming (HCLP). We propose an alternative approach for describing preferred criteria in CLP as a problem of relational optimization (RO). In this approach the programmer defines a preference relation which indicates when a solution is better than another solution. We study several schemes based on pruning for optimizing an objective function, and we show how these schemes can be generalized to handle preference relations defined by CLP programs, while preserving a straightforward logical semantics. Further we show on some examples that the greater flexibility of the relational optimization scheme is not at the cost of efficiency.

Keywords: preference, constraint logic programming, optimization, hierarchical constraint logic programming.

1 Introduction

Very often when we have a constraint satisfaction problem we are not simply interested in finding any solution, but we have a notion of best solutions to the problem. The difficulty of describing what constitutes the best solutions to a problem depends on the choice of preferred criteria. When we state a problem with constraints we are presented with the problem of how to state these preferences. If we do not indicate the preferences at all we are left with a large number of possible solutions which we must then sort to find the most interesting solutions. Alternatively if we insist that all the preferences we may have for a solution be satisfied then there may be no solution satisfying all our preferences. We see that it is important to use these preferences actively in such a way that all but the least important preferences are satisfied, this scenario is particularly true for decision support systems.

One approach consists of specifying all intended constraints as either mandatory or as preferences, with an associated label indicating the strength of the preference, this creates a constraint hierarchy. In [18] Hierarchical Constraint

Logic Programming (HCLP) was proposed as a general approach for integrating constraint hierarchies in CLP. The HCLP scheme gives rise to a family of languages each member parameterized both by the domain of constraints, as in the CLP scheme, and by the comparator of solutions. The comparator of solutions defines the solutions to a constraint hierarchy.

We propose a different approach to integrating preferences in CLP based on relational optimization. In this approach the programmer defines a preference relation which indicates when a solution is better than another solution. We study several schemes based on pruning for optimizing an objective function, and we show how these schemes can be generalized to handle preference relations defined by CLP programs, while preserving a straightforward logical semantics. Further we show on some examples that the greater flexibility of the relational optimization scheme is not at the cost of efficiency.

We have used the current implementation to resolve a real-life preference problem consisting of allocating bands of frequencies from a radio spectre to a set of networks [7]. In this problem, preferences between solutions depend on several conflictual criteria.

In the rest of the paper we review the basic ideas of constraint hierarchies before presenting HCLP languages. We then present relational optimization. We compare the advantages and disadvantages of the two schemes before evaluating the performances of the relational optimization approach.

2 HCLP approach

HCLP languages were presented in [1] [17] [18] as an approach to express preferences in CLP [8]. These languages integrate constraint hierarchies and logic programming.

2.1 Constraint Hierarchies

A labelled constraint is a constraint c labelled with a strength s and is written c@s. The strengths range from 0 to n where n is the number of non-required levels. Level 0 corresponds to the required constraints and the other levels to non-required constraints. The higher the level the weaker the constraint is.

A constraint hierarchy is a multiset of labelled constraints. Given a constraint hierarchy $\mathcal{H}$, $\mathcal{H}_0$ denotes the required constraints in $\mathcal{H}$ with their labels removed. The sets $\mathcal{H}_1, \mathcal{H}_2, \ldots$ are defined in the same way for levels $1, 2, \ldots$ respectively.

A valuation, θ, for a constraint hierarchy, $\mathcal{H}$, is a mapping of the free variables of the hierarchy to values in the domains of the variables that satisfies all the required constraints, and is noted $\mathcal{H}\theta$. Different valuations are compared using a comparator to find those valuations which relax only the least important constraints. A comparator, $>$, is a relation between valuations of a constraint hierarchy. A solution to a constraint hierarchy is a valuation such that there does not exist a valuation better than it.

$$\theta \text{ is a solution to a constraint hierarchy } \mathcal{H} \text{ iff } \neg\exists\sigma\ \mathcal{H}\sigma > \mathcal{H}\theta$$

There are many choices for a comparator. It must, however, be a relation that is transitive, anti-reflexive and respects the hierarchy. A comparator is said to respect the hierarchy if when there exists a valuation that satisfies all the constraints upto a level k, then any solution must satisfy all the constraints upto level k.

In [1] comparators are divided into two classes: local and global. Comparators are defined by the choice of an error function, e, for a constraint and in the case of global comparators an error function, g, for the combined errors of the constraints.

The error function for a constraint, c, under a substitution θ is a function which returns a non-negative number indicating how nearly the substitution satisfies the constraint. If the substitution satisfies the constraint the function always returns zero. The function g combines the errors of all the constraints at a given level.

locally-better$(\theta, \sigma, \mathcal{H}) \equiv$
  $\exists k > 0$ such that
    $\forall i \; \epsilon \; 1, \ldots, k-1 \; \forall p \; \epsilon \; \mathcal{H}_i \; e(p\theta) = e(p\sigma)$
    $\wedge \; \exists q \; \epsilon \; \mathcal{H}_k \; e(q\theta) < e(q\sigma)$
    $\wedge \; \forall r \; \epsilon \; \mathcal{H}_k \; e(r\theta) \leq e(r\sigma)$

globally-better$(\theta, \sigma, \mathcal{H}, g) \equiv$
  $\exists k > 0$ such that
    $\forall i \; \epsilon \; 1, \ldots, k-1 \; g(\theta, \mathcal{H}_i) = g(\sigma, \mathcal{H}_i)$
    $\wedge \; g(\theta, \mathcal{H}_k) < g(\sigma, \mathcal{H}_k)$

We illustrate the "unsatisfied count better" comparator ($>_{ucb}$), which is a global comparator. For this comparator e is defined as $e(c\theta) = 0$ iff $c\theta$ holds and $e(c\theta) = 1$ otherwise, and the combining function is $g(\theta, \mathcal{H}_i) = \sum_{c \epsilon \mathcal{H}_i} e(c\theta)$.

2.2 HCLP

In HCLP, programs are just like CLP programs where constraints have an additional strength annotation (no annotation means required constraint). In CLP a CSLD derivation returns a set of constraints, in the case of HCLP this set is a constraint hierarchy $\mathcal{H}$. For example the program below builds two constraint hierarchies: {X=1,X>3 @ 1} and {X=5,X>3 @ 1}.

```
p(X) :- X>3 @ 1, q(X).
q(1).
q(5).
```

So the solutions to p(X) are X=1 and X=5, and these solutions are not comparable, because the theory presented in [1] and summarized above does not make comparisons between solutions from different constraint hierarchies. We would like, however, to obtain as an answer only the second, because it satisfies more constraints in its respective hierarchy than the first answer. These sort of problems have led to an extension of the notion of comparators to allow comparisons for solutions arising from different constraint hierarchies. These extended comparators were first presented in [17], and are called inter-hierarchy comparators.

2.3 Inter-hierarchy comparators

In this extended model we consider not just one but several hierarchies, and the definitions are simply extended to cover this more general case. The tree of CSLD derivations associated for a given goal returns the set of constraint hierarchies Δ. A valuation to a set of constraint hierarchies Δ is a mapping of the free variables to values in the domains of the variables that satisfies all the required constraints of at least one constraint hierarchy. A valuation θ for a constraint hierarchy $\mathcal{H}$ written $\theta_{\mathcal{H}}$, is globally-better than a valuation $\sigma_{\mathcal{J}}$ is defined below, where $\mathcal{H}, J \epsilon \Delta$

```
globally-better($\theta_{\mathcal{H}}, \sigma_{\mathcal{J}}, \Delta, g$) $\equiv$
  $\exists k > 0$ such that
    $\forall i \in 1, \ldots, k-1 \; g(\theta, \mathcal{H}_i) = g(\sigma, \mathcal{J}_i)$
    $\wedge \; g(\theta, \mathcal{H}_k) < g(\sigma, \mathcal{J}_k)$
```

We remark that local comparators have not been extended to this framework, because these comparators consider each constraint in the hierarchy individually.

2.4 Nonprimitive constraints

As stated in [17] conjunction and disjunction of constraints in HCLP, as opposed to CLP, cannot be accomplished without explicitly using disjunction and conjunction connectives. [17] introduces nonprimitive constraints that allow conjunction and disjunction of constraints by the explicit use of connectives. A nonprimitive constraint is a labelled constraint built from primitive constraints and logical connectives.

So the previous example may be rewritten using a nonprimitive constraint as shown below:

```
p(X) :- X>3 @ 1, (X=1 \/ X=5).}
```

This creates one constraint hierarchy {X=1∨X=5,X>3 @ 1} which has solution X=5.

This allows us to achieve the effect of inter-hierarchy comparison by allowing the disjunction of primitive constraints as illustrated in the previous example. As indicated in [17], it is not always possible to avoid inter-hierarchy comparison in this way. Consider the following example.

```
p(X) :- X=1, X=<0 @ 1.
p(X) :- X=2.
```

For this example, two constraint hierarchies are created: {X=1,X=<0 @ 1} and {X=2}. A locally-predicate-better comparator will return X=1 and X=2 as solutions. A global comparator will return only X=2.

If we try to apply a disjunction as in a previous example, the only possibility is the following program:

```
p(X) :- (X=1 \/ X=2), X=<0 @ 1.}
```

This creates a constraint hierarchy {(X=1∨X=2),X=<0 @ 1}. Unfortunately, we obtain the same solutions X=1 and X=2 for a locally-predicate-better comparator. Worse still, certain global comparators (for example, weighted-sum-better, see [17] for more details) will only return X=1, as a solution.

3 Optimization approach

We have seen how preferences are handled in *HCLP*. We now present the approach that we use for handling preferences as a *CLP* optimization problem. This section relies on the branch and bound like procedures presented in [16] and [12] for the optimization of an objective function in CLP, and generalized in [5] in accordance with a logical semantics based on negation. We review these results in the context of preference constraints, and extend them for the handling of program-defined preference relations.

3.1 Syntax

The idea is to express the preference relation among solutions inside the language, that is by a CLP program. For this we introduce an optimization higher-order predicate, *rel_opt*, whose first argument is a goal and whose second argument is a preference relation. Intuitively a higher-order atom of the form

```
rel_opt(goal(X),better)
```

is true if X is an optimal solution to $goal(X)$, that is $goal(X)$ is true and there does not exist a Y such that $goal(Y)$ and $better(Y, X)$ are true:

$$rel\_opt(goal(X), better) \Leftrightarrow goal(X) \wedge \neg\exists Y(goal(Y) \wedge better(Y, X))$$

An *OCLP program* is a CLP program that may contain occurrences of the optimization predicate in rule bodies.

The preference predicate *better* is supposed to be defined in the CLP program by a set of rules which state when a solution X to *goal(X)* must be preferred to a solution Y. It is in the *better* rules that all preference criteria are described. Clearly the relation *better* should be anti-reflexive for the *rel_opt* goals be satisfiable. We shall assume also transitivity although this is not always necessary in our framework. Note that in [2] the same question is discussed in the framework of preference logic.

Going back to the example of the previous section:

```
p(X) :- X>3 @ 1, q(X).
q(1).
q(5).
```

The meaning of that HCLP program under the extended model with inter-hierarchy comparisons is given by the set of derivations to the goal `p(X)` where the derivations which satisfy `X>3` subsume the others. This example can be expressed declaratively with an OCLP program:

```
better(X,Y) :- X > 3, Y =< 3.
p(X) :- rel_opt(q(X),better).
q(1).
q(5).
```

An important particular case is when it is defined by a primitive constraint on a measure of satisfiability of preference constraints. In that case a preference relation expresses an *objective function*. The underlying algebra is a strict total order $(\mathcal{A}, <)$ and the definition of the preference relation is of the form

```
better(Y,X) :- f(Y)<f(X).
```

for some objective function f that acts as a measure of satisfiability of the preference constraints. The total order $<$ can be the lexicographic or any extension of total orders on preference criteria.

In general the predicate *better* need not be deterministic, it may be recursive. Note also that relational optimization metapredicates allow to express preferences not only at *top level* but also inside a program. This is crucial for the handling of composite systems with preference constraints defined locally for some components of the system. Recursion through relational optimization predicates is also supported by the syntax and the semantics presented here, but we shall not investigate that case any further in this paper.

3.2 Declarative semantics

The simplest and most general way to provide a declarative semantics to CLP programs with relational optimization is to read an atom

```
rel_opt(goal(X),better)
```

as an abbreviation for the formula

$$goal(X) \wedge \neg(\exists Y goal(Y) \wedge better(Y, X))$$

and then take a suitable semantics for CLP programs with negation as the semantics of the original program with preference constraints.

The natural semantics to employ in this context is the semantics of Kunen [9] for normal logic programs, based on Fitting operator and generalized by Stuckey [15] to CLP programs. That semantics is defined as the set of three-valued logical consequences (denoted by $\models_3$) of the Clark's completion of the program.

The usual strong 3-valued interpretations of the connectives and quantifiers are assumed, except for the connective $a \leftrightarrow b$ which is two-valued: t if a and b have the same truth value ($\{f, t, u\}$), and f otherwise. The Clark's completion, $comp(P, \mathcal{A})$, of a CLP program P over a structure A supposed to be presented by a theory $th(\mathcal{A})$, is defined as the conjunction of $th(\mathcal{A})$ together with the formulae of the form

$$\forall x_1, ..., x_n \; p(x_1, ..., x_n) \leftrightarrow \bigvee_i \exists Y^i (x_1 = t_1^i \wedge ... \wedge x_n = t_n^i \wedge B^i)$$

obtained for each predicate symbol p (of arity n) by collecting the clauses headed by p in P, $\{p(t_1^i, ..., t_n^i) \leftarrow B^i\}$ where the set of local variables is denoted by Y^i, or of the form

$$\forall x_1, ..., x_n \; \neg p(x_1, ..., x_n)$$

if p does not appear in any head in P.

Now a constraint c is a correct answer to a goal G if

$$comp(P, \mathcal{A}) \models_3 \forall (c \rightarrow G) \wedge \exists c$$

The answer no is correct if

$$comp(P, \mathcal{A}) \models_3 \neg\exists(G).$$

Constructive negation, as introduced by Chan for logic programs, and generalized by Stuckey to CLP programs, provides a correct and complete operational semantics w.r.t. the declarative semantics [15]. In the following subsections we examine simpler practical operational semantics by distinguishing the different cases where preferences are expressed either by an objective function or a program defined predicate, and where preferences are handled either on the top level goal only, or recursively in the subgoals of the CLP program.

3.3 Operational semantics for the optimization of an objective function

Optimization of the top level goal In this section we examine the case studied in [12] where preferences on the solutions of the top level goal are encoded by the minimization of an objective function. The underlying structure is a strict total order $(\mathcal{A}, <)$ and the predicate *better(X, Y)* is a constraint of the form $f(Y) < f(X)$ for some objective function f. We assume that the language of constraints is closed by minimization of the objective function, that is we can decide whether

$$min_{c(X)} f(X)$$

has a solution or not for any constraint c of the language (or at least for those constraints returned by the successful derivations of the top level goal to optimize).

In this framework, preferences can be used actively to reduce the search space by a simple pruning mechanism.

Procedure 1 *The computation of the answers to the goal* $minimize(G(X), f(X))$ *proceeds as follows:*

1. *a CSLD derivation tree for* $G(X)$ *is developed,*
2. *once a successful derivation is found, say with answer constraint* $c(X)$*, then the optimal cost*
$$v = min_{c(X)} f(X)$$
is computed. If v *doesn't exist then return the answer no, otherwise the derivation tree is pruned by adding the constraint*
$$f(X) \leq v$$
to the nodes, the pruned tree is further developed by iterating step 2,
3. *once the derivation tree gets finite, the constraints on the successful derivations after pruning, if any, represent the optimal solutions to the goal, otherwise the computed answer is no.*

Proposition 1. *(Correctness and completeness) The answers computed by procedure 1 are correct. Furthermore, if* $c(X)$ *is a correct answer to the goal* $G(X) \wedge \neg \exists Y (f(Y) < f(X) \wedge G(Y))$ *then there exist computed answers* $c_1, ..., c_n$ *such that* $\mathcal{A} \models c \rightarrow \exists Y_1 c_1 \vee ... \vee \exists Y_n c_n$ *where the* Y_i*'s are the variables in* c_i *not in* c*.*

OCLP programs with objective functions When general OCLP programs and goals are considered, preferences need be handled not only at top level but also in recursive subgoals. Procedure 1 can be generalized to this purpose by adding to the standard CSLD resolution rule a special rule for optimization predicates based on pruning [5]:

Procedure 2 *The computed answers to a goal* G *and an OCLP program* P *are those defined by a CSLD derivation tree for* G *in which the successors of a node labeled by a goal of the form*
$$c|minimize(G(X), f(X)), A_1, ..., A_n$$
where the optimization atom is the selected atom are formed by:

1. *developing a CSLD derivation tree for* $minimize(c|G(X), f(X))$ *with procedure 1, return no successor in case of failure, otherwise let* v *be the optimal cost found on a successful derivation,*
2. *check that the goal* $f(Y) < v|G(Y)$ *finitely fails, otherwise return no successor,*
3. *return the successor goals* $c \wedge c_i|A_1, ..., A_n$ *for each successful derivation with answer constraint* c_i *obtained in step 1.*

The rationale of procedure 2 is that a solution to a constraint c and an atom $minimize(G(X), f(X))$ is a solution to $minimize(c|G(X), f(X))$ such that $\neg \exists Y (f(Y) < f(X) \wedge G(Y))$. Note however that procedure 2 may loop forever in step 1 trying to solve `min(c|G(X),f(X))`, whilst the optimiality condition for the goal `c|min(G(X),f(X))` is unsatisfiable. This prevents a completeness result for no answers. For instance with the program

```
p(0).
p(X) :- X>1, p(X).
```

and the goal `X>1|minimize(p(X),X)` the sequential pruning procedure loops forever on the goal `minimize(X>1|p(X), X)`, whilst

$$comp(P, \mathcal{A}) \models_3 \neg\exists X(X > 1 \wedge p(X) \wedge \neg\exists Y(Y < X \wedge p(Y)).$$

Completeness on success holds however for *2-level OCLP programs*, i.e. for programs such that there is no dependency between atoms through more than one minimization predicate. Completeness holds also under finite evaluation assumptions w.r.t. the goal passed to optimization predicates. This is the case for instance when these goals are defined by hierarchical programs.

Proposition 2. *The answers computed by procedure 2 are correct. Procedure 2 is complete on success for 2-level programs, and complete both on success and failure for hierarchical programs.*

Note that a concurrent pruning procedure complete both on success and failures can be derived for OCLP programs from the scheme for constructive negation based on pruning described in [6].

3.4 Operational semantics for relational optimization

The main difficulty with the previous scheme for handling preference constraints is that all preferences need to be explicitly encoded by an objective function. We show here how this difficulty can be overcomed by expressing the preference among solutions with a CLP program and by generalizing the previous pruning procedures to handle program-defined preference relations.

Ground answers In this subsection we assume that the goals passed as arguments of optimization predicates return always ground answers. This is typically the case in CLP(FD) applications where the solutions to a combinatorial problem are naturally a valuation of the unknowns, and where enumeration is mixed with constraint propagation in order to palliate the incompleteness of the constraint solvers. However to be efficient the optimization of the preference relation must be performed by a pruning mechanism that cut the branches of the derivation tree before the variables get instanciated.

This behavior can be achieved by a simple generalization of the previous procedures. We assume that the program-defined preference relation *better(X, Y)* is anti-reflexive and transitive, and that the complement relation *notbetter(X, Y)* is also defined in the program. If the preference relation is total, *notbetter(X, Y)* is simply the relation Y better or equal to X.

First procedure 1 is generalized by replacing the pruning based on cost constraints, by a pruning mechanism based on a *better* goal:

Procedure 3 *The answers to the goal rel_opt*$(G(X), better)$ *are computed by:*

1. developing a CSLD derivation tree for $G(X)$,
2. once a successful derivation is found, say with (ground) answer substitution X_i, then the derivation tree is transformed by adding the goal

$$notbetter(X_i, X)$$

to the nodes, the tree is further developed by iterating step 2,
3. once the derivation tree gets finite, the constraints on the successful derivations after pruning, if any, represent the optimal solutions to the goal, otherwise the computed answer is no.

Proposition 3. *Procedure 3 is correct and complete.*

Procedure 2 can be generalized in the same way for solving recursively a goal of the form

$$c|rel\_opt(G(X), better), A_1, ..., A_n$$

by checking in step 2 for each successful answer X_i to $rel\_opt(c|G(X), better)$, that the goal *better(Y,Xi), G(Y)* finitely fails. The correctness and completeness results generalize in a straightforward way to this scheme.

Procedure 3 is the procedure we use in the examples reported in the section on performance evaluation.

General case The general case of relational optimization leads to the general problem of constructive negation in CLP languages. The original scheme of [15] for constructive negation is not practical as it necessitates taking the disjunctive normal form of an entire frontier in the derivation tree of a negated subgoal, at each resolution step with a negative subgoal. In [3] a new scheme called intensional negation is proposed as a compiled version of constructive negation. In that scheme the normalization process is done once and for all at compile time. The idea is to derive from the definition of a predicate p in $comp(P, \mathcal{A})$ a positive program for $\neg p$ that contains complex subgoals formed with universally quantified disjunctions. For instance the resulting program on the previous example is:

```
better¬(X,Y) :- X =< 3.
better¬(X,Y) :- Y > 3.
q¬(X) :- X ≠ 1, X ≠ 5.
p(X) :- q(X), ∀Y (q¬(Y) ∨ better¬(Y,X)).
q(1).
q(5).
```

Complex subgoals involve universally quantified disjunctions. The derivation rule for complex subgoals is defined in a way similar to constructive negation and requires generally to compute entire frontiers in a CSLD derivation trees. In the previous example the absence of recursion makes it possible to perform these resolution steps at compile time (by unfolding). The result for f is the clause `p(X) :- q(X), ∀Y ((Y ≠ 1, Y ≠ 5) ∨ Y =< 3 ∨ X > 3).` which simplifies to a simple form

```
p(X) :- q(X), X>3.
```

In general however the process of collecting a complete frontier in the derivation tree of the selected subgoal must be performed at each resolution step with a complex subgoal.

Finally note that the scheme for constructive negation based on pruning described in [6], eliminates the need to consider complex subgoals with explicit quantifiers, and should be of practical value for handling program defined preferences in a very general setting.

4 Comparison

We have presented two different approaches for integrating preferences inside the CLP framework. We now discuss and contrast the interesting aspects and applications of the two approaches.

There is a fundamental difference between HCLP and relational optimization. Relational optimization (RO) describes preferences on solutions to a goal regardless of the derivations taken to find these solutions. HCLP attaches preference constraints to rules, so the criteria of preference among solutions may be different from one derivation to another.

We argue that RO reflects the intended semantics of obtaining the best solutions to a problem independently of the method used to calculate them. This is reflected by the straightforward declarative semantics of the relational optimization schemes. One major inconvenience of previous optimization schemes with objective functions is that in general there is no natural choice for such a function. We have seen that RO allows the expression of the preference relation among solutions by a CLP program providing a greater flexibility. In the next section this is illustrated on a multicriteria frequency allocation problem. Furthermore the ability to put the RO predicate inside rule bodies, allows the handling of different preference subproblems locally in the program.

On the other hand HCLP provides a way to express preferences locally inside rules without stating a global preference on solutions when this is still too difficult and unweildy to express easily.

For a given hierarchy of constraints there is an encoding of HC in RO and vice versa. For instance the set of solutions to a constraint hierarchy H, defined [1] by:

$S = \{\theta | \theta \in S_0 \wedge \forall \sigma \in S_0$
$\neg(G([E(H_1(\sigma)), \ldots, E(H_n(\sigma))]) <_g G([E(H_1(\theta)), \ldots, E(H_n(\theta))]))\}$
where $S_0 = \{\theta | \forall c \in H_0 \; e(c\theta) = 0\}$, is the set of solutions to the required constraints of H, can be defined as well as the set of X satisfying

$s_0(X) \wedge \forall Y s_0(Y) \rightarrow \neg(G([E(H_1(Y)), \ldots, E(H_n(Y))]) <_g G([E(H_1(X)), \ldots, E(H_n(X))]))$

[1] In this more general definition from [18], E applies e to every element in a set of constraints and G is a generalization of g that is applied to error sequences.

where s_0 denotes the membership predicate in S_0. So S can be defined by the relational optimization goal `rel_opt(s0(X),better)` with the program

```
better(X,Y) :- G([E(H1(X)),...,E(Hn(X))<g G([E(H1(Y)),...,E(Hn(Y))
s0(X) :- C(X)
```

where $C(X)$ is the set of required constraints in H.

Implementing the relational optimization schemes can be efficiently done by writing a meta-interpreter on top of of CLP system. RO inherits on one hand from the objective function schemes based on pruning, and on the other hand from the work on negation in logic programming for which increasingly powerful inference rules have been given. This should be contrasted with the ad hoc implementation of the HCLP scheme.

We have implemented the RO scheme on top of the CLP system Meta(F) [10, 4]. In the following section we evaluate the performance of relational optimization to resolve preference problems.

5 Performance evaluation

This section is intended to show some performances of the the relational optimization approach. More particularly, we have evaluated the relational optimization on a time tabling problem and on an industrial frequency allocation problem which has multiple preference criteria. The time tabling problem is taken from [13].

5.1 A Time-tabling Problem

This problem is a typical resource allocation problem. There are the typical required constraints, and there are four hierarchic levels of prefered criteria. The most important prefered criterion (level 1) is that lessons of the same subject should not be on consecutive days. The level 2 prefered criterion is that lessons of the same subject should not be more than two days apart. The level 3 criterion is that any subject with less than two lessons per week should not have any of its lessons on a Monday. The final and weakest preference is that lessons of the same subject should take place at the same hour in the week.

We have used the relational optimization scheme to state the same preferences among solutions as those described by the HCLP program presented in [14] using the $>_{ucb}$ comparator which is defined earlier.

The first table presents a loose timetable problem and the second a compact problem, which is more difficult than the first because the number of hours available per day is reduced by one hour. The first column of the tables indicates the level of the preference that we try to satisfy, thus max level three means that we try to maximize the number of preferences respected of level one, two and three, but not four.

The cpu time in seconds required for the specific implementation of HCLP ($>_{ucb}$,FD) which has been developed on the Incremental Hierarchical Constraint Solver (IHCS) [14]. These times are given for the prototype implementation written in C using a YAP prolog software platform running on a NeXT Station 68040. We present the cpu time given in seconds using our current relational optimization implementation on a Sun Sparc Station IPX (an IPX can be considered 30% faster than the NeXT for a standard integer performance test (MIPS)). We also report the number of preferred constraints not respected by the solutions found, _ indicates that the preference constraints of this level are not being used.

Max.	Number of	IHCS	RO	Not Respected			
level	constraints	Time	Time	@1	@2	@3	@4
0	356	1.80s	0.78s	_	_	_	_
1	+21 = 377	1.86s	1.23s	2	_	_	_
2	+21 = 398	1.98s	1.38s	2	1	_	_
3	+11 = 409	1.98s	1.61s	2	1	1	_
4	+21 = 430	2.38s	2.12s	2	1	1	0

Table 1. *results for loose time-tables*

Max.	Number of	IHCS	RO	Not Respected			
level	constraints	Time	Time	@1	@2	@3	@4
0	356	1.80s	0.75s	_	_	_	_
1	+21 = 377	1.86s	1.28s	2	_	_	_
2	+21 = 398	2.28s	1.51s	2	2	_	_
3	+11 = 409	4.63s	1.63s	2	2	2	_
4	+21 = 430	2m43.53s	2.05s	2	2	2	0

Table 2. *results for compact time-tables*

For both tables and both solvers we see that, as is expected, there is a deterioration in performance with the introduction of supplementary levels of preferred constraints. The deterioration in computation time for the relational optimization approach is less sensitive to the number of levels of preferred constraints to be treated and the number of constraints relaxed than the IHCS solver. These timings show that our current RO implementation competes well with the HCLP implementation of [14].

5.2 An application to multiple criteria preference problems

The comparative approach introduced by the use of rules to define the relationships between solutions (*better*), greatly enlarges the techniques that we may use for multiple criteria optimization [7]. Sequential elimination methods [11] may be used in this framework. Sequential elimination methods are based on comparing solutions on the basis of the values of their attributes and eliminating inferior solutions. The comparative nature of these methods makes it easier to represent expert knowledge and give a process account of expert reasoning.

We have applied these principles to the allocation of frequency bands from a radio spectre to a group of networks. The radio spectre has two to three thousand distinct frequencies. The system allocates frequency bands for several hundred networks, by partitioning them into subsets containing about twenty networks, and allocating frequencies to these subsets.

The allocation of frequencies to networks is constrained to respect forbidden frequency constraints, network frequency constraints, and interference constraints. In a typical problem for the current system about a thousand constraints are required to state all the properties an allocation must respect.

- Some of the frequencies of the radio spectre are forbidden to all networks, in general this represents around thirty percent of the spectre.
- The network frequency constraints limit the number of frequencies available per network and their possible distributions. A network is constrained to have between thirty and two hundred and fifty frequencies, which may be split into at most two separate bands of frequencies.
- The interference constraints guarantee that when two networks are close the bands of frequencies that they are allocated will be sufficiently distant to avoid interference. The separation of bands of frequencies for two networks is a function of the degree of proximity and the frequencies allocated. Three degrees of proximity are identified: no interference, low interference and high interference. The higher the frequency allocated the greater the distance that must be respected to avoid interference.

The difficulty of the problem lies in the definition of what constitutes a good placement for the bands of frequencies. Here several criteria were exposed as important attributes of a possible frequency allocation:

- the number of frequencies per network should be maximized (C1);
- the number of networks with two frequency bands should be maximized (C2);
- the separation between frequency bands on the same network should be maximized (C3);
- the distribution of frequencies among networks should be equitable (C4).

A good allocation would have all networks with two well separated bands, containing a large number of frequencies, and with all networks having a similar number of frequencies.

A first approximation for multi-criteria optimization was to try a strict lexicographic order (C1,C2,C3,C4) among criteria. This was not satisfactory, according to the experts, so we introduced thresholds, to modify the underlying lexicographic order. Furhter we combined the criteria C1 and C4 to form a new criterion C5 and similarly C2 and C3 to form C6.

The optimization relation we retained for the frequency allocation problem is the following:

- (C6,C5) is the lexicographic order until the two band threshold has been reached for C6;

– after the two band threshold has been reached the lexicographic order is (C5,C6).

The rules we used to implement the allocation preferences in the frequency allocation system are given below. The predicate `solution_criteria` measures the value of a solution on each of the four criteria defined above, and `lexicographic` is a strict lexicographic order.

```
better(Sol1,Sol2):-
  solution_info(Sol1,Sol2,C51,C61,C52,C62,TBT),
  TBT > C62,
  lexicographic([C61,C51],[C62,C52]).

better(Sol1,Sol2):-
   solution_info(Sol1,Sol2,C51,C61,C52,C62,TBT),
   C62 >= TBT,
   C61 >= TBT,
   lexicographic([C51,C61],[C52,C62]).

solution_info(Sol1,Sol2,C51,C61,C52,C62,TBT):-
   two_band_threshold(TBT),
   solution_criteria(Sol1,C11,C21,C31,C41),
   solution_criteria(Sol2,C12,C22,C32,C42),
   combine12(C11,C21,C51),
   combine34(C31,C41,C61),
   combine12(C11,C21,C51),
   combine34(C31,C41,C61).

lexicographic([H0|_],[H1|_]):-
   H0>H1,!.
lexicographic([H0|T0],[H1|T1]):-
   H0=H1,
   lexicographic(T0,T1).
```

In the current system the optimization process does not finish but is stopped when satisfactory solutions have been found. In fact good solutions, already better than those found by experts, are found after one to two minutes.

6 Conclusion

Hierarchical constraints and relational optimization are two ways for stating preferences inside a CLP framework. The relational optimization scheme is founded on the semantics of negation in CLP languages, and provides a uniform treatment of preference constraints and multi-criteria optimization. Our first experiments have shown that this greater generality was not at the cost of efficiency and that real-life applications could be tackled with this approach. We are currently exploring the integration of the scheme for constructive negation by pruning of [6] in our implementation.

References

1. Alan Borning, Michael J. Maher, Amy Martindale, and Molly Wilson. Constraint hierarchies and logic programming. In G. Levi and M. Martelli, editors, *Logic Programming: Proceedings of the 6th International Conference (Lisbon)*, pages 149–164. MIT Press, June 1989.
2. A. Brown Jr et al. Constraint optimization using preference logics: A new role for modal logic. In *PPCP 93 First Workshop on Priniciples and Practice of Constraint Programming*, Newport, USA, April 1993.
3. P. Bruscoli, F. Levi, G. Levi, and M.C. Meo. Intensional negation in clp. In *GULP 93*, Italy, 1993.
4. P. Codognet, F. Fages, and T. Sola. A metalevel compiler of clp(fd) and its combination with intelligent backtracking. In A. Colmerauer, editor, *Constraints logic programming: collected papers.* MIT Press, 1993.
5. F Fages. On the semantics of optimization predicates in clp languages. In *FSTTCS Foundations of software technology and theoretical computer science*, Bombay, India, December 1993. Springer-Verlag.
6. F Fages. Constructive negation by pruning. In *Ecole de Printemps d'Informatique Théorique*, Chatillon sur Seine, France, May 1994. Working paper.
7. Julian Fowler, Henri Dieudonné, and Jean Jourdan. How to handle multiple criteria optimization in constraint logic programming. Technical report, Thomson CSF/LCR, November 1993. 93-3.
8. Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages, Munich, Germany*, pages 111–119. ACM, January 1987.
9. K. Kunen. Negation in logic programming. *Journal of Logic Programming*, 4(3):289–308, 1987.
10. R. Lissajoux. Meta(f) 3.40 users' manual. Technical Report LACS 93-2, Thomson-CSF, June 1993.
11. Kenneth R. MacCrimmon. *An overview of multiple objective decision making.*
12. M.J. Maher and P.J. Stuckey. Expanding query power in constraint logic programming languages. In E. Lusk and R. Overbeek, editors, *NACLP89*, pages 20–36. MITP, oct 1989.
13. F Menezes et al. An incremental hierachical constraint solver. In *PPCP 93 First Workshop on Priniciples and Practice of Constraint Programming*, Newport, USA, April 1993.
14. F Menezes et al. An incremental hierachical constraint solver applied to a timetabling problem. In *Proceedings of Avignon 93*, May 1993.
15. P. Stuckey. Constructive negation for constraint logic programming. In *LICS'91*, 1991.
16. P. Van Hentenryck. *Constraint Satisfaction in Logic Programming.* Logic Programming Series. MIT Press, Cambridge, MA, 1989.
17. M. Wilson and A. Borning. Extending hierarchical constraint logic programming: Nonmonotonicity and inter-hierarchy comparison. In E. Lusk and R. Overbeek, editors, *NACLP89*, pages 3–19. MITP, oct 1989.

18. Molly Wilson and Alan Borning. Hierarchical constraint logic programming. *Journal of Logic Programming*, 16(3 and 4):277–318, July and August 1993.