Transformations between HCLP and PCSP

Michael Jampel[*], Jean-Marie Jacquet[†] David Gilbert, Sebastian Hunt

mja,jmj@info.fundp.ac.be
Institut d'Informatique
F.U.N.D.P.
rue Grandgagnage 21
B-5000 Namur, Belgium

drg,seb@cs.city.ac.uk
Dept. of Computer Science
City University
Northampton Square
London EC1V 0HB, U.K.

February 13, 1996

Abstract

We present a general methodology for transforming between HCLP and PCSP in both directions. HCLP and PCSP each have advantages when modelling problems, and each have advantages when implementing models and solving them. Using the work presented in this paper, the appropriate paradigm can be used for each of these steps, with a meaning-preserving transformation in between if necessary.

Keywords: Hierarchical CLP, Partial Constraint Satisfaction, transformation

Contact information:

Michael Jampel
mja@info.fundp.ac.be
(address as above)
phone: +32 81 72.49.87
fax: +32 81 72.49.67

[*]Michael Jampel has been funded by City University during his PhD, and is currently funded by the European Community under the TMR scheme.

[†]Jean-Marie Jacquet is supported by the Belgian National Fund for Scientific Research as a Research Associate. Part of this work was carried out in the context of the INTAS project 93-1702 "Efficient Symbolic Computing."

1 Introduction

The Hierarchical Constraint Logic Programming (HCLP) scheme of Borning, Wilson, and others [3, 10, 12] greatly extends the expressibility of the general CLP scheme [7]. A semantics has been defined for HCLP [10, 11] and some instances of it have been implemented [8, 10].

The Partial Constraint Satisfaction (PCSP) scheme of Freuder and Wallace [4, 6] is an interesting extension of CSP, which allows the relaxation and optimisation of problems. Extensive empirical studies have been made of some of its instances [6].

There is a widespread view that some link exists between particular HCLP problems and particular PCSP problems, but no general method of transforming one into the other is present in the literature. General frameworks have been developed of which HCLP and PCSP are particular instances [1, 9], but they do not provide a method for transforming between them. In this paper we present a completely general method for finding the HCLP equivalent of any PCSP problem, and vice versa.

Our motivation is mainly methodological, to allow the use of whichever paradigm is appropriate for specification, even if the other one is more appropriate for execution. But we also have a more theoretical motivation, namely to show the relationship between the two formalism's orthogonal approaches to over-constrained systems (OCSs). The two orthogonal approaches are as follows: HCLP reorganises the *structure* of an over-constrained problem, by specifying relationships between constraints; PCSP keeps a flat structure to the problem, but changes the *meaning* of the individual constraints (by adding elements to the domain).

The structure of this paper is as follows. In Section 2 we introduce HCLP and PCSP. We then make some preliminary remarks in Section 3. We discuss transforming HCLP into PCSP in Section 4, including an example in Section 4.2. Then Section 5 discusses transformations from PCSP into HCLP. Pseudo-code illustrating both these transformations can be found in Section 6. Finally in Section 7 we present some conclusions and also mention further work.

2 Background

2.1 Hierarchical Constraint Logic Programming

A good introduction to HCLP can be found in Molly Wilson's PhD thesis [10, chapter 4] or in the early reference [3]; here is a brief overview. CLP can be extended to a Hierarchical CLP scheme including both 'hard' and 'soft' constraints. The HCLP scheme is parameterised not only by the constraint domain $\mathcal{D}$ but also by the 'comparator' $\mathcal{C}$, which is used to compare and select from the different ways of satisfying the soft constraints.

The strengths of the different constraints are indicated by a non-negative integer label. Constraints labelled with a zero are *required* (hard), while constraints labelled j for some $j > 0$ are optional (soft), and are preferred over those labelled k, where $k > j$. (A program can include a list of symbolic names, such as *required*, *strongly-preferred*, etc., for the strength labels, which will be mapped to the natural numbers by the interpreter. If the strength label on a constraint is omitted, it is assumed to be *required*.)

The constraint store σ (a set) is partitioned into the set of required constraints S_0 and the set of optional ones S_i. The solution set for the whole hierarchy is a subset of the solution set of S_0, such that no other solution could be 'better', i.e. for all levels up to k, S_k is completely satisfied, and for level S_{k+1} this solution is better than all others, in terms of some comparator. Backtracking and incomparable hierarchies give rise to multiple possible solution sets, each a subset of the solution to S_0.

'Better' is defined with respect to some comparator [12]. The key notion is that a comparator is a function from a solution of a set of constraints to a sequence of numbers, which are then ordered lexicographically; the first element of the sequence measures how well the solution satisfies the required constraints, the second how well the strongest optional constraints are satisfied, etc.; the earlier in the order, the better that solution is.

See Section 3.3 for more detail on the particular aspects of HCLP involved in the transformations.

2.2 Partial Constraint Satisfaction Problems

Freuder has developed a theory of Partial Constraint Satisfaction Problems (PCSPs) to weaken systems of constraints which have no solutions, or for which finding a solution would take too long [4, 6]. PCSP is formalised as containing three components

$$\langle (P, U), (PS, \leq), (M, (N, S)) \rangle$$

where P is a constraint satisfaction problem (CSP), U is a set of 'universes' i.e. a set of potential values for each of the variables in P, $(PS, \leq)$ is a problem space with PS a set of problems and $\leq$ a partial order over problems, M is a distance function over the problem space, and (N, S) are necessary and sufficient bounds on the distance between the given problem P and some solvable member of the problem space PS.

A solution to a PCSP is a problem P' from the problem space and its solution, where the distance between P and P' is less than N. If the distance between P and P' is minimal, then this solution is optimal.

2.2.1 Constraint Satisfaction Problems

The definition of a constraint satisfaction problem is well known: it consists of a pair $\langle V, C \rangle$ where V is a set of variables, each with a domain (extension), and C is a set of constraints[1]. Solving a CSP involves finding one value from the domain of each variable such that all the constraints are satisfied simultaneously. Generally the CSP world restricts itself to considering binary constraints over variables with finite domains. A constraint c between two variables x and y can be denoted c_{xy}.

(The domains of the variables in V are usually considered as unary constraints, but in order to simplify the presentation in [4] they are represented as binary constraints between a variable and itself. The value v is in the domain of a variable x if c_{xx} contains (v, v). In fact, unless there are elements in the domain of a variable which do not appear in any constraint, it is redundant to state individual variable domains explicitly: we can always reconstruct them by saying that $U :: \{i \mid (i, j) \in C_{UV} \text{ or } (k, i) \in C_{WU}, \text{ for all } j, k, V, W\}$.)

[1]Constraints are relations over the variables in V. In CSPs, they are usually treated extensionally, i.e. a binary constraint is just considered as a set of pairs.

2.2.2 The problem space

A problem space PS is a partially-ordered set of CSPs where the order $\leq$ is defined as follows (*sols(P)* denotes the set of solutions to a CSP called P):

$$P_1 \leq P_2 \text{ iff } sols(P_1) \supseteq sols(P_2)$$

Note that the ordering is over *problems*, but defined in terms of *solutions*. The problem space for a PCSP must contain the original problem P, which can provide the maximal element in the order, for standard problem spaces.(In the most general case, PS can in fact contain Q such that $P \leq Q$ or such that P and Q are incomparable. But if we take the conjunction of all the constraints in all the problems in PS and create a single problem R, then R will definitely be the greatest element in the order.) If P has no solutions, then *sols(P)* $= \{\}$, which is a subset of all other sets.

The obvious problem space to explore when trying to weaken a problem is the collection of all problems Q such that $Q \leq P$, but it may also be useful to consider only some of these Qs, i.e. those problems which have been weakened in a particular way which makes sense in the context of the system that we are trying to model.

2.2.3 Weakening a problem

There are four ways to weaken a CSP: (a) enlarging the domain of a variable, (b) enlarging the domain of a constraint, (c) removing a variable, and (d) removing a constraint. Consider example Z above: if none of your shirts match your shoes, you could buy new shoes (variable domain enlargement / augmentation), you could decide that certain shoes do, after all, go with a certain shirt (constraint augmentation), you could decide not to wear shoes at all (variable removal), or you could ignore clashes between shoes and shirts (constraint removal). (As a comparison with these four methods, in HCLP we could decide that the constraint that shirts match shoes is simply not very important.)

Freuder shows in [4] that these can all be considered in terms of (b) above i.e. enlarging constraint domains (adding extra pairs to the relation which defines the constraint). (a) As we have already decided to consider the domains of variables as binary constraints c_{xx}, domain enlargement can clearly be achieved by constraint augmentation. (d) Enlarging a constraint c_{xy} until it equals $x \times y$ (the cartesian product of the domains) has the same effect as removing it altogether. (c) Removing all the constraints on a variable achieves the aim of removing the variable itself.

2.2.4 The distance function

Different distance functions are possible, but one obvious one is derived from the partial order on the problem space. If $M(P, P')$ equals the number of solutions not shared by P and P', then when $P' \leq P$ the distance function measures how many solutions have been added by the relaxation of P. Another distance function is a count of the number of constraint values not shared by P and P', and yet others could be based on HCLP-like strength labels. Freuder suggests that a distance function may be used which will tend to find weakened problems with certain properties, for example one whose constraint graph has certain structural properties (for example, see [5]).

3 Transformation: preliminary remarks

HCLP and PCSP are not identical in scope, therefore it is impossible to transform all of HCLP into PCSP. This work presented in the rest of this paper is *complete* i.e. we present transformations for every single aspect which it makes sense to transform. First of all, however, we discuss those parts of HCLP which are outside the scope of PCSP, and make other preliminary remarks.

3.1 Differences which will not be transformed away

Firstly[2], CLP in general defines a class of programming languages, which place constraint solving in a logic programming framework, whereas CSP defines a set of problems, techniques, and algorithms. We could embed PCSP in a logic programming framework, and then a comparison with HCLP would make sense, or we can ignore the programming language aspects of HCLP, and compare the resulting theory of 'constraint hierarchies' with PCSP. In this section we will consider the latter approach, i.e. when we say 'HCLP' we really mean 'constraint hierarchies'.

Secondly, CSP techniques are always defined with finite domains whereas the CLP framework extends to continuous domains such as the real numbers. We will only attempt to transform HCLP(FD); however, we *will* transform metric comparators as well as predicate ones. (Metric comparators required a notion of 'distance' between points in the domain, but there is no reason why this distance cannot be discrete.)

Finally, in HCLP the required constraints are special; the difference between required and strong constraints is richer than the difference between, say, strong and weak. PCSP does not have this special class of required constraints. This is discussed further in the next section.

3.2 PCSP with distinguished required constraints

In Section 2.2, we presented the standard formalisation of PCSPs as $\langle (P, U), (PS, \leq), (M, (N, S))\rangle$. We can modify this to allow us to denote a subset of the constraints in P as 'required', giving a theory which can be called $\mathrm{P_RCSP}$ (our additions in italics):

$$\langle (P, R, U), (PS, \leq), (M, (N, S))\rangle$$

where P is a constraint satisfaction problem, *$R \subseteq P$ is a set of constraints*, U is a set of 'universes' i.e. a set of potential values for each of the variables in P, $(PS, \leq)$ is a problem space with PS a set of problems *each of which contains all the constraints in R*, and $\leq$ a partial order over problems, M is a 'distance function' on the problem space, and (N, S) are necessary and sufficient bounds on the distance between the given problem P and some solvable member of the problem space PS. A solution to a PCSP is a problem P' from the problem space and its solution, where the distance between P and P' is less than N, and *where all the constraints in R are satisfied.* If the distance between P and P' is minimal, then this solution is optimal.

[2]The three points mentioned in this section are reasonably straightforward, but have not been explicitly made in any publication. They were mentioned to one of the authors by Borning [Private Communication], but we were already aware of them independently.

In Section 2.2 we noted that Freuder states that the obvious problem space to explore when trying to weaken a problem is the collection of all problems Q such that $Q \leq P$, but we also noted that it may be useful to consider only some of these Qs, i.e. those problems which have been weakened in a particular way which makes sense in the context of the system that we are trying to model [5]. Therefore we note that P_RCSP can be considered simply as selecting those Qs which satisfy all the constraints in R.

One way to select the appropriate part of the problem space is to choose a distance function which gives an infinitely large distance for all other parts. If distance functions are generally denoted by μ, from now on we will assume the existence of a particular function μ_∞, usually parameterised by a set of required constraints σ, which defines a distance of zero to any problem which satisfies all the constraints in σ, and a distance of infinity to all other problems. If T is some arbitrary problem drawn from the problem space, then

$$\mu_{\infty(\sigma)} = \begin{cases} 0, & \text{if } \sigma \subseteq T \\ \infty, & \text{otherwise} \end{cases}$$

$\mu_{\infty(\sigma_r)}$ will be the first element of the sequence of functions $\mu = [\mu_r, \mu_s, \mu_w, \ldots]$ parameterised by the constraints at each level of the hierarchy. For example, if the comparator used is UCB, then $\mu = [\mu_{\infty(\sigma_r)}, \mu_{UCB(\sigma_s)}, \mu_{UCB(\sigma_w)}, \ldots]$.

The main conclusion of this section is that we can deal with the issue of required constraints in a straightforward and localised manner. Therefore, perhaps surprisingly, in the rest of this paper we do not really need to emphasise the difference between PCSP and P_RCSP.

3.3 Characterisation of HCLP and PCSP

In this section we present those aspects which are relevant for the transformation process. The relevant aspects for HCLP are

$$\langle \mathbf{H} = (\mathbf{H_0}, \mathbf{H_1}, \mathbf{H_2}, \ldots]), \mathcal{C} = (e, \mathbf{E}, g) \rangle$$

where $\mathbf{H}$ is a hierarchy of constraints, made up of all the required constraints $\mathbf{H_0}$, the strongly preferred constraints $\mathbf{H_1}$, weaker preferences $\mathbf{H_2}$ etc. The comparator $\mathcal{C}$ is used to compare different solutions; it is made up of an error function e which calculates the error of a possible solution with respect to one constraint, $\mathbf{E}$ which simply maps e pointwise over all the constraints in one level of the hierarchy $\mathbf{H_i}$, and a combining function g which combines the elements of the sequence produced by $\mathbf{E}$, resulting in a score for that solution with respect to all the constraints at that level of the hierarchy. For example g might be 'max', or 'sum', or 'least squares'. The resulting sequence of errors $[r, s, w, \ldots]$ giving the errors with respect to each level if the hierarchy, are used to order different possible solutions lexicographically. The lowest element in the order indicates the best solution.

PCSP is formalised as a triple $\langle (P, U), (PS, \leq), (M, (N, S)) \rangle$, but we need only consider certain elements of it as follows: P is a constraint satisfaction problem, and M is a distance function which selects the consistent problem 'nearest' to P.

When transforming HCLP into PCSP, we will take all the constraints in $\mathbf{H}$ without their strength labels as being P. We will use the strength label information and the comparator to construct the appropriate distance function.

When transforming PCSP into HCLP, the constraints in the hierarchy will just be the constraints in P, and the distance function will be used to define their strength labels (i.e. which of the $\mathbf{H_i}$ should contain each constraint) and the comparator $\mathcal{C}$.

In the case of the standard PCSP distance function, all the constraints from P must be placed in the same non-required level of the hierarchy, but it does not matter which one is used. Arbitrarily, we choose to label them 'strong' and so put them in $\mathbf{H_1}$.

4 Transforming HCLP into PCSP

4.1 Creating the distance function

The base problem (P) is *all* the constraints in the hierarchy, without their strength labels. U, PS, and (N, S) remain as they would for an original PCSP based on P. (By 'original PCSP' we mean one written down by a user, as opposed to one created by automatically transforming an HCLP problem.)

The distance function will be calculated from a combination of the HCLP comparator and the particular hierarchy of labelled constraints, and the hierarchy will lead to it being stratified into a lexicographic order. The distance function μ derived from a hierarchy with n levels will be stratified into n parts, whose results will be ordered lexicographically (i.e. it will not calculate a single distance of the relaxed CSP from P). Each relaxation (each problem drawn from the problem space PS) will be annotated with a sequence $[d_0, d_1, d_2, \ldots, d_{n-1}]$ each element of which is calculated by the respective distance function in $\mu = [\mu_0, \ldots, \mu_{n-1}]$. (The required level is formally called level 0, the strongest non-required level is 1, down to $n-1$ for the weakest level.) For example, in the case of a hierarchy containing only required, strong and weak constraints, each candidate problem will be annotated with a sequence $[r, s, w]$, where r is the distance according to μ_r, the part of the distance function derived from the required constraints, s is the distance according to μ_s, the part of the distance function derived from the strong constraints, and w is the weak distance, calculated by μ_w. We then order the various relaxations according to the lexicographical order of their sequences.

The distance function calculates the distance of one of the problems T in the problem space PS from the 'ideal' set of constraints which would have distance zero (i.e. completely satisfy all the constraints in the original problem). In fact, as the original constraints might be inconsistent, it is possible that no such ideal set exists.

Let us define $sols(T)$ to be the set of solutions to T. We required T to be consistent, and so $sols(T)$ will never be empty. Each member of $sols(T)$ is a valuation, i.e. an assignment of a value from its domain to each variable in T. We can calculate how well a particular valuation satisfies the constraints σ using the machinery developed by Borning and Wilson for HCLP.

T may have more than one solution, and hence may give rise to more than one valuation, therefore we define the distance of T from σ to be the *maximum* of distances of each of the valuations in T. This is necessary because HCLP's comparators take as input the set of original constraints and a single valuation / possible solution. The output is the score for that particular valuation, which can then be used to place that valuation in an order. In PCSP, however, distance functions create an order over sets of constraints; a set of constraints can have many solutions, and so we have to choose the score of one of them. We choose the worst (largest) score, i.e. this set of constraints can never give an answer with a score worse than x. For example, if T is said to be a distance of 2 from σ, that means that any solution of T is a distance of *at most* 2 from σ.

Therefore, using some HCLP terminology including denoting a general comparator by $\mathcal{C}$ (defined in terms of g, e, and $\mathbf{E}$), the PCSP distance function defined in terms of the set σ of constraints from one optional level of the hierarchy is:

$$\mu_{\mathcal{C}(\sigma)}(T) = \max\{g(\mathbf{E}(\sigma\tau) \mid \tau \in \mathit{sols}(T)\}$$

In other words, we treat all the constraints in σ as a sequence, apply a particular valuation τ to each of them, calculate the error for each member of the sequence, combine the errors using g, and then take the maximum of the errors for all the τ and treat it as the error for T.

The various distance functions, each parameterised by the constraints from a different level of the hierarchy, will lead to results which are lexicographically ordered, just as in HCLP. The main difference between standard HCLP and our work is that we interpose the step of taking the maximum error for each of the valuations in T between the application of g and placing in an order.

In the case of UCB, $g(\mathbf{v}) = \sum_{i=1}^{|\mathbf{v}|} v_i$ and $e = e_p$ is the simple predicate error function which returns 0 for each constraint in σ which is consistent with τ, and 1 for each inconsistent constraint [2, 12]. $\mathbf{E}$ is e raised over sequences, i.e. its input is a sequence of constraints, and its output in this case is a sequence of 0's and 1's. g then adds all these individual errors. Least-squares-better (LSB) has a more complicated $e = e_d$, which measures the error as a 'distance' in a metric space. g then sums the squares of these errors:

$$\begin{aligned}
\mu_{UCB(\sigma)}(T) &= \max\left\{\sum_{c\in\sigma} e_p(c,\tau) \mid \tau \in \mathit{sols}(T)\right\} \\
\mu_{LSB(\sigma)}(T) &= \max\left\{\sum_{c\in\sigma} e_d(c,\tau)^2 \mid \tau \in \mathit{sols}(T)\right\}
\end{aligned}$$

See Section 6 for pseudo-code for transforming HCLP into PCSP.

4.2 Example

In this section we present an example of an over-constrained system and its specification and solution in HCLP, and then showing its transformation into PCSP.

Consider the problem of choosing matching clothes (example adapted from Freuder and Wallace [6]). A robot wishes to wear a shirt, some shoes, and some trousers, and wants them all to match each other. There are various choices for the different items and various constraints between them. We can easily model this using three finite domain variables with a number of binary constraints between them. If we use the letter S to denote the variable for shirts, then we can use F for shoes (footwear) and T for trousers. The domain of the shirt variable will be $S :: \{r, w\}$ for red and white respectively, and similarly shoes and trousers will have domains $F :: \{c, s\}$ for cordovans and sneakers, and $T :: \{b, d, g\}$, for blue, denim, and grey. A constraint that shirts must match footwear will be denoted C_{SF}, and so on. Then, using Freuder and Wallace's assumptions about which clothes go with which, the complete problem can be expressed formally as follows (we will call this model Z):

$$S :: \{r, w\},\ F :: \{c, s\},\ T :: \{b, d, g\}$$
$$C_{ST} :: \{(r, g), (w, b), (w, d)\},\ C_{FT} :: \{(s, d), (c, g)\},\ C_{SF} :: \{(w, c)\}$$

This problem is over-constrained; it has no solutions. We can see this by choosing the red shirt, and tracing the implications of this choice. We must choose the grey trousers, which forces us to choose the cordovans as footwear. But according to C_{SF}, the cordovans only go with the white shirt. Contradiction. We can trace the effects of choosing the white shirt in the same way, also arriving at a contradiction. Therefore we need to consider some way of relaxing or weakening the problem until solutions can be found.

4.2.1 Example in HCLP

Let us use HCLP strength labels to indicate our assumption that, say, shirts and trousers are more important than footwear, and let us choose the *unsatisfied-count-better* (UCB) comparator:

strong C_{ST}, weak C_{FT}, weak C_{SF}

The solutions to this hierarchy will equal the solutions to the two equally acceptable relaxed problems (C_{ST}, C_{FT}) and (C_{ST}, C_{SF}) which are, in the variable order (S, F, T), $\{(r, c, g), (w, s, d)\}$ and $\{(w, c, b), (w, c, d)\}$ respectively.

4.2.2 HCLP formulation transformed into PCSP

The base set of constraints for the PCSP formulation will be all the constraints from the HCLP version, without strength labels. The distance function will be in two parts $\mu = [\mu_{UCB(C_{ST})}, \mu_{UCB(C_{FT},C_{SF})}]$, one of which measures the relaxation of the strong constraint, and another for the weak level of the hierarchy. The order will be the lexicographic order over the sequences of integers $[s, w]$ produced by μ.

UCB and $\mu_{UCB(\sigma)}$ are elsewhere in this paper, as is the notion of constraint augmentation. Here it suffices to say that the best solutions, i.e. those earliest in the order created by the distance function, will be the sets of constraints $\{C_{ST}, C_{FT}, C'_{SF}\}$, $\{C_{ST}, C_{FT}, C''_{SF}\}$, $\{C_{ST}, C'_{FT}, C_{SF}\}$, $\{C_{ST}, C''_{FT}, C_{SF}\}$, where $C'_{SF} = \{(w, c), (\mathbf{r}, \mathbf{c})\}$, i.e. C_{SF} augmented with the extra tuple (r, c), and the other three solutions also contain one augmented constraint ($C''_{SF} = \{(w, c), (\mathbf{w}, \mathbf{s})\}$, $C'_{FT} = \{(s, d), (c, g), (\mathbf{c}, \mathbf{b})\}$, $C''_{FT} = \{(s, d), (c, g), (\mathbf{c}, \mathbf{d})\}$). The solutions from these four sets of constraints, in variable order (S, F, T), are $\{(r, c, g)\}$, $\{(w, s, d)\}$, $\{(w, c, b)\}$, and $\{(w, c, d)\}$, identical to the HCLP solutions.

5 Transforming PCSP into HCLP

5.1 Transforming the standard PCSP distance function

To transform PCSP with the standard distance function into HCLP, we take the constraints in P and give them all the same arbitrary non-required strength label, say 'strong'. Thus they will be placed in $\mathbf{H_1}$. Then we use the HCLP comparator unsatisfied-count-better (UCB). We claim that this is the correct comparator to use, i.e. we claim that the solutions calculated by HCLP using UCB are the same as those in PCSP, and the particular solutions which are best according to PCSP will also be best according to UCB. (The intuition is as follows: the number of unsatisfied constraints counted by UCB is the same as the number of constraints which would need a single domain augmentation to create a consistent CSP, thus UCB measures an equivalent distance to that measured in PCSP.)

Certain combinations of augmented constraints in the PCSP formulation, which duplicate solutions found at a closer distance, will not appear in the HCLP answer, but all the solutions to these combinations *will* appear. (Here is an analogy: if the list of PCSP solutions, in order from best to worst, is $[a, b, c, a, d, a, e]$, the list of HCLP solutions may be $[a, b, c, d, e]$. So although the lists are not equal, the fact that a should be chosen before b or d is present in both representations.)

See Section 6 for pseudo-code for transforming PCSP into HCLP.

5.1.1 Detailed defence of choice of UCB

This section contains a detailed defence of our choice of UCB as the comparator to use in HCLP when transforming from PCSP. It addresses one possible key objection, but does not affect the presentation in subsequent sections of the paper.

Consider those PCSP weakenings which involve more than one augmentation of a single constraint. We claim that the following complaint about our choice of UCB is unjustified: "UCB will just detect that a constraint had been violated by a valuation. It wouldn't detect that two different augmentations would be necessary for the constraint not to be violated." It is incoherent because two augmentations can never be necessary for a *single* constraint not to be violated. Two augmentations to a single constraint might, however, lead to an additional two or more solutions, but we can ignore this situation due to the following claim:

Claim: the additional solutions caused by $n \geq 2$ augmentations of a single constraint can be completely separated into n classes, each of which contains solutions caused by only one of the n augmentations. The CSPs represented by these singly-augmented constraints will all appear in the partial order induced by the distance function, and they will all appear earlier than the CSP containing the n-augmented constraint. Therefore, no solutions will be lost by ignoring all multiply-augmented constraints. Therefore, the fact that UCB only picks out those solutions which violate the smallest number of singly-augmented constraints, does not change the set of solutions computed. (All that would happen is that two solutions s_1 and s_2 will separately appear as, say, the equal-best solutions to the hierarchy, but their union will fail to appear as a second-best or third-best solution.)

Example: Let A' denote the constraint A with one extra tuple added to its domain, in the usual manner. Usually there will be more than one way to augment A; these alternatives may be indicated by A'_1, A'_2, etc. Let A'' generally denote two augmentations to A, and specifically $A''_{1,2}$ denote that the two augmentations are equivalent to $A'_1 \cup A'_2$. Then our claim is that all the solutions to the CSP $\{A''_{1,2}, B''_{3,4}, C''_{5,6}\}$ are present in the union of the solution sets $\{A'_1, B'_3, C'_5\} \cup \{A'_2, B'_3, C'_5\} \cup \{A'_1, B'_4, C'_5\} \cup \{A'_2, B'_4, C'_5\} \cup \ldots$ In other words, we can ignore multiple augmentations of a single constraint.

Intuition: Consider the CSP as a graph, with each variable represented by a node and each constraint represented by an edge. The tuples which make up the constraint are labels for the edges. A solution to the CSP is a path through every edge in the graph, consistent with the labels. If we add a label to an edge, we are increasing by one the number of paths between the two nodes connected by that edge[3]. If instead we added a different label, we would again increase the number of paths between these two nodes by one. It is intuitively clear that adding these two labels simultaneously will add precisely two paths between the two nodes: any path can only take account of one of the two labels on the edge. We could have arrived at the same set of total paths through the graph by taking two copies of the original graph, adding one new label to each of them, finding the new paths caused by this single extra label, and then eventually taking the union of the two sets of paths.

Proof: Consider various binary constraints over different pairs selected from n variables X_1, X_2, X_3, etc. We can define the *expansion* C^*_{ij} of each constraint C_{ij}, which originally related X_i and X_j, to a set of n-tuples by creating a tuple for each element of the cartesian product of the variables not originally involved in the constraint:

$$C^*_{ij} = \{(v_1, v_2, \ldots, v_i, v_j, \ldots, v_n) \mid (v_i, v_j) \in C_{ij}, (v_k, k \neq i, k \neq j) \in \mathrm{dom}(X_k)\}$$

Example: if X has domain $\{a, b\}$, Y has domain $\{c, d\}$, and Z has domain $\{e, f\}$, and if $A_{XY} = \{(a, c), (b, d)\}$, then $A^*_{XY} = \{(a, c, e), (a, c, f), (b, d, e), (b, d, f)\}$.

It is clear that the solution to a CSP is precisely the intersection of the expanded versions of each of its constraints. Thus instead of considering the solution of the set of constraints $\{A_{XY}, B_{YZ}, C_{XZ}\}$, we can just consider $A^*_{XY} \cap B^*_{YZ} \cap C^*_{XZ}$.

If we add one pair to the domain of one of the constraints in a CSP, it is equivalent to adding a set of n-tuples to the domain of that constraint's expanded version, where the other places in the tuple are filled with all possible combinations of elements from the domains of all the other variables. Continuing with the example, let us assume, without loss of generality, that we have augmented constraint B. This leads to adding a set of n-tuples to B^*; let us call this set of additional tuples R. We can imagine adding a different pair to B which would lead to adding a different set to B^*, say R'. If we add both pairs to B at the same time, it is clear[4] that we must add $R \cup R'$ to B^*.

[3]The number of paths through the entire graph may increase by more than one. If there are k paths leading into the start node of the edge under consideration, and l paths leading away from the end node, then adding a path between the two nodes may increase the number of paths through the entire graph by up to kl.

[4]If it is not clear, consider the following: if the first pair added to constraint C_{ij} is (α, β) (giving, say, ${}'_1C$) and the second pair is (γ, δ) (giving ${}'_2C$), and the doubly-augmented constraint is called ${}''C$,

then
$$\begin{aligned} {}'_1C^*_{ij} &= C^*_{ij} \cup \{(v_1, v_2, \ldots, \alpha, \beta, \ldots, v_n) \mid (v_k, k \neq i, k \neq j) \in \mathrm{dom}(X_k)\} \\ {}'_2C^*_{ij} &= C^*_{ij} \cup \{(v_1, v_2, \ldots, \gamma, \delta, \ldots, v_n) \mid (v_k, k \neq i, k \neq j) \in \mathrm{dom}(X_k)\} \end{aligned}$$

and
$$\begin{aligned} {}''C^*_{ij} &= C^*_{ij} \cup \{(v_1, v_2, \ldots, v_i, v_j, \ldots, v_n) \mid (v_i, v_j) \in \{(\alpha, \beta), (\gamma, \delta)\}, \\ &\qquad (v_k, k \neq i, k \neq j) \in \mathrm{dom}(X_k)\} \end{aligned}$$

then clearly
$${}''C^*_{ij} = {}'_1C^*_{ij} \cup {}'_2C^*_{ij}$$

Our claim is that we can ignore CSPs where one constraint has been multiply augmented; all their solutions will be present in the union of the solutions to CSPs with singly-augmented constraints. This is equivalent to claiming

$$A^* \cap (B^* \cup \underline{(R \cup R')}) \cap C^* = (A^* \cap (B^* \cup \underline{R}) \cap C^*) \underline{\cup} (A^* \cap (B^* \cup \underline{R'}) \cap C^*)$$

The proof is a straightforward exercise in the use of the distributivity laws of set theory ($J \cup (K \cap L) = (J \cup K) \cap (J \cup L)$ and its dual), with one use of the idempotence of set union ($K \cup K = K$).

Therefore, using UCB as our comparator in the automatically generated HCLP version of a PCSP is acceptable. So our transformation from PCSP to HCLP holds.

5.2 Transforming non-standard distance functions

We have shown above how to transform problems using the standard PCSP distance function into HCLP. We now consider three other possibilities, firstly where all the variables and constraints are treated equally by the distance function but the distance is not defined as minimum augmentation, secondly where some of the *variables* in the problem are highlighted, and finally where some of the *constraints* are highlighted.

5.2.1 Non-specific (homogeneous) distance functions

All the constraints are put at the 'strong' level of the hierarchy resulting from the transformation. The combining function embodied by the distance function must be transformed into an HCLP-like comparator, specifically into an error function for each constraint and a combining function which combines the errors at each level.

5.2.2 Distance functions which prefer a subset of the variables

In general CSPs are considered in terms of binary constraints. The theory can be extended, but complications are introduced. CLP, on the other hand, is indifferent to the arity of constraints. Therefore, if a PCSP problem has some kind of cost function which selects solutions which minimise the value of some function of (some of) the variables, we can simply treat it as another constraint. If the use of the cost function is expressed in the usual way ("Do not violate any constraints in order to minimise the function") then it can be labelled 'weak', while all the constraints in the original PCSP are labelled 'strong.' If it is acceptable to violate constraints in order to minimise the function, then the inverse strength labelling can be used.

5.2.3 Distance functions which prefer a subset of the constraints

This possibility can be transformed into HCLP in a very straightforward manner: the preferred constraints are labelled 'strong', while the others are labelled 'weak'. If there are multiple subsets with some order over them, then clearly more HCLP strength levels can be used.

6 Pseudo-code

In Figure 1 we present logic-programming-style pseudo-code which transforms HCLP into PCSP in the manner described in Section 4 of this paper. Figure 2 presents similar pseudo-code for the PCSP to HCLP transformation (Section 5). We can also write a single procedure, called say `transform-HP`, which includes transformations in both directions thus showing the relational nature of our transformations; we have kept them separate here for clarity of presentation. Using the relational version, we state the main claim of this paper as a query in logic programming terms in Figure 3. This Figure assumes the existence of two procedures, each of which interfaces to a standard implementation of HCLP and PCSP respectively. The HCLP procedure requires a collection of labelled constraints and a comparator as input. The PCSP procedure has as inputs a collection of unlabelled constraints and a distance function. Figure 3 claims that if we do not have an implementation of HCLP, we can replace a call to it by the two calls `transform-HP, PCSP`, and vice-versa. So we can specify in one of the two paradigms and solve in the other, whenever it is necessary or desirable.

```
transform-HCLP-PCSP( (LabelledConstraints, Comparator),
                           (Constraints, DF) ) :-
    partition-constraints( LabelledConstraints,
                           (Required, Strong, Weak, ...  ),
    remove-labels( (Required, Strong, Weak, ...),
                           (UL-Required, UL-Strong, UL-Weak, ...)  ),

       % DF = Distance function.
       % The type of DF is determined by the $e$ and $g$ functions
       % (which are determined by the choice of comparator), and
       % and also parameterised by the different levels of constraints
    distance-function-0( (Required, Comparator), DF-R(UL-Required) ),
    distance-function-1( (Strong, Comparator), DF-S(UL-Strong) ),
    distance-function-2( (Weak, Comparator), DF-W(UL-Weak) ),
       ...
    collect-distance-functions( (DF-R(UL-Required), DF-S(UL-Strong),
                           DF-W(UL-Weak)...), DF ),
    collect-constraints( (UL-Required, UL-Strong, UL-Weak,...),
                           Constraints ).
```

Figure 1: HCLP into PCSP

7 Conclusions and further work

7.1 Conclusions

We have developed a general methodology for transforming between HCLP and PCSP. We have clarified various issues, and provided a proof of correctness. We have shown

```
transform-PCSP-HCLP( (Constraints, DF(Type,SpecialCons)),
                           (LabelledConstraints, Comparator) ) :-
       % if Type = standard, then UCB comparator will be chosen, etc.
       % if no constraints are highlighted by the distance function,
       % then SpecialCons will be empty and all constraints are ‘strong’

    partition-constraints( (Constraints, DF(Type,SpecialCons))
                           (UL-Strong, UL-Weak, ...  ),
    add-labels( (UL-Strong, UL-Weak, ...), (Strong, Weak, ...)  ),

    create-comparator(Type, Comparator)
    collect-constraints( (Strong, Weak, ...), LabelledConstraints ).
```

Figure 2: PCSP into HCLP

```
?-  HCLP( (LabelledConstraints, Comparator), HCLP-Solutions ),
    transform-HP( (LabelledConstraints, Comparator),
                            (Constraints, DF) )
    PCSP( (Constraints, DF), PCSP-Solutions ),
    equiv( HCLP-Solutions, PCSP-Solutions ).
```

Figure 3: Equivalence

that strength labels, associated with constraints in HCLP, contain information which is necessary to define the global distance function in PCSP.

HCLP and PCSP each have advantages when modelling problems, and each have advantages when implementing models and solving them. Using the work presented in this paper, the appropriate paradigm can be used for each of these steps, with a meaning-preserving transformation in between if necessary.

7.2 Further work

We would like to investigate issues of algorithmic complexity within the two paradigms.

Acknowledgements

Thanks to Alan Borning, Thomas Schiex, Rob Scott, and Roland Yap for many helpful discussions about CLP.

Thanks to City University for funding Michael Jampel's PhD for three years, and to the European Community for a TMR grant. Thanks to the Belgian National Fund for Scientific Research, and to the European Community INTAS project 93-1702, for funding Jean-Marie Jacquet's research.

References

[1] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Constraint Solving over Semirings. In *IJCAI'95*, Montreal, August 1995.

[2] Alan Borning, Bjorn Freeman-Benson, and Molly Wilson. Constraint Hierarchies. *Lisp and Symbolic Computation*, 5:223–270, 1992.

[3] Alan Borning, Michael Maher, Amy Martindale, and Molly Wilson. Constraint Hierarchies and Logic Programming. In *ICLP'89* Lisbon, Portugal, June 1989.

[4] Eugene Freuder. Partial Constraint Satisfaction. In *IJCAI'89*, August 1989.

[5] Eugene Freuder. Exploiting Structure in Constraint Satisfaction Problems. In *Constraint Programming: Proceedings 1993 NATO ASI Parnu, Estonia*, pages 54–79. Springer, 1994.

[6] Eugene Freuder and Richard Wallace. Partial Constraint Satisfaction. *Artificial Intelligence*, 58:21–70, 1992.

[7] Joxan Jaffar and Jean-Louis Lassez. Constraint Logic Programming. In *POPL'87* Munich, 1987.

[8] Francisco Menezes, Pedro Barahona, and Philippe Codognet. An Incremental Hierarchical Constraint Solver. In Paris Kanellakis, Jean-Louis Lassez, and Vijay Saraswat, editors, *PPCP'93: First Workshop on Principles and Practice of Constraint Programming*, Providence RI, 1993.

[9] Thomas Schiex, Hélène Fargier, and Gerard Verfaillie. Valued Constraint Satisfaction Problems: Hard and Easy Problems. In *IJCAI'95*, Montreal, August 1995.

[10] Molly Wilson. *Hierarchical Constraint Logic Programming.* PhD thesis, University of Washington, Seattle, May 1993. (Also available as Technical Report 93-05-01).

[11] Molly Wilson and Alan Borning. Extending Hierarchical Constraint Logic Programming: Nonmonotonicity and Inter-Hierarchy Comparison. In *NACLP'89* Cleveland, Ohio, 1989.

[12] Molly Wilson and Alan Borning. Hierarchical Constraint Logic Programming. *Journal of Logic Programming*, 16(3):277–318, July 1993.