Constraint Hierarchies in Constraint Logic Programming Languages

Mouhssine Bouzoubaa

Departmenet of Optimisation SINTEF
Postboks 124, Blindern, Norway
mbo@math.sintef.no

Abstract

Houria is an incremental solver that proposes a new implementation of constraint hierarchies. Houria uses local propagation to maintain sets of required and preferential constraints. It represents constraints between variables by sets of short procedures (methods) and incrementally re-satisfies the set of constraints when individual constraints are added and removed. The criteria of comparison used in this solver are global. They allow the comparison of valuations, which are not comparable by local criteria used in existing solvers. The solution found by Houria satisfies more constraints than the one produced by other solvers and that for the same over-constrained problems while respecting the semantics of the hierarchy. We also propose an efficient algorithm that integrates the Houria solver in the CLP paradigm.

Keywords: Constraint-Based Reasoning, Logic and Constraint Programming.

1 Introduction

Local propagation is an efficient constraint satisfaction algorithm that takes advantage of potential locality of constraint systems [11]. The solver in Garnet [9] is based on local propagation and handles one-way constraints. A one-way constraint always outputs a value to a certain variable. However, one-way constraints are often insufficient because they cannot change dependencies among variables. Multi-way constraints are proposed in [10]. A multi-way constraint has multiple methods for one constraint. A system of multi-way constraints is solved as follows : one method is selected from each constraint, a solution graph is generated out of the system so that the graph has no conflicts and no cycles then, local propagation is applied to the solution graph. Multi-way constraints also embody the problem that output variables are not determined uniquely and it would easily result in over-constrained systems. Borning et al. in [5, 6] proposed constraint hierarchies to cope with the problem of over-constrained systems. A constraint hierarchy is a system of constraints with hierarchical strengths.

If the system is over-constrained, it is solved so that there are as many satisfied strong constraints as possible. For example the constraints $v_2 = 1$ and $v_2 = 2$ are in conflict. However, if $v_2 = 1$, $v_2 = 2$ are respectively associated with strengths *strong* and *weak*, the constraint system is solved by satisfying only $v_2 = 1$.

The existing solvers (Blue, DeltaBlue, SkyBlue [12, 8, 4]) for over-constrained systems, are based on local criteria: valuation comparison. According to these local criteria, a lot of valuations might not be comparable (i.e. two different valuations satisfying two disjoint constraint sets are not comparable) and regarding the semantics of the hierarchy, the best valuation can not be obtained. In order to be able to find a valuation which satisfies more constraints than the one produced by these solvers. We propose an efficient solver based on a global criterion to solve a constraint hierarchy. Thus, we are able to compare the valuation sets that are incomparable by a local criterion. A constraint hierarchy consists of a set of constraints, each labeled as either hard or soft at some strength, and each soft constraint is weighted by a real-valued weight. Existing hierarchical solvers require that all constraints within a specific level in the hierarchy have the same weight. Houria surmounts this restriction by accommodating some extended definitions. It produces solution graphs, and applies local propagation to them. The planning (for obtaining the best solution graph(s)) and execution (for obtaining the valuation) time is acceptable for some random over-constrained problems. We present also a second algorithm based on Houria for comparisons between the hierarchies that arise from alternate rule choices in a program writing in CLP language.

2 Theory of Functional Constraint Hierarchies

Each functional constraint has a set of methods that can be invoked to satisfy the constraint. For example, the constraint $v_1 = v_2 - v_3$ has three methods: $v_1 \leftarrow v_2 - v_3$ (*i.e.* calculates the value of v_1 from the values of v_2 and v_3), $v_2 \leftarrow v_1 + v_3$ and $v_3 \leftarrow v_2 - v_1$.

In [3], a functional constraint hierarchy is a triplet (V, D, C) defined by a set of n variables: $V = \{v_1, v_2, ..., v_n\}$. Each variable v_i ranges over a domain d_i . The set of domains d_i is noted by $D : D = \{d_1, d_2, ...d_n\}$ and a constraint system C. A constraint is an n-ary relation among a subset of V. Each constraint has some methods. A method uses some of the constraint variables as inputs called "antecedents" and computes the remainder as outputs called "consequents". A method may only be executed when all of its input variables are determined by at least one other constraint, and none of its outputs have been determined by other constraints. Each constraint is associated with a strength i where: $0 \leq i \leq m$. Strength 0 represents the strength of required constraints, C is partitioned into sets $C_0, C_1, C_2, ..., C_m$ where C_i contains the constraints with strength i . C_1 contains the most strongly preferred constraints. C_2 the next weaker level, and so forth through C_m, where m is the number of distinct non-required strengths. Solutions S to a constraint hierarchy are defined as a set of valuations, each valuation in S must be such that it satisfies all constraints in C_0. In addition, we desire each valuation in S to be such that it satisfies the non-required

constraints $C_1, C_2, ..., C_m$ as well as possible, with respect to their relative strengths.

To formally define this set of solutions, we first present the set S_0 of valuations such that each valuation in this set satisfy C_0. Then, using S_0, we present the desired set S by eliminating all potential valuations that are worse than some other potential valuations using the comparator predicate better. (Sat is a boolean predicate, its value is true when the valuation θ satisfies every constraint in C_0, otherwise its value is false).

$$S_0 = \{\theta : Sat(\theta, C_0)\} \; and \; S = \{\theta : \theta \in S_0 \wedge \forall \eta \in S_0 \neg better_C(\eta, \theta)\}.$$

Many alternate definitions for comparators are given in [3, 5, 6], "*better*" is irreflexive and transitive. However, *better* will not provide a total ordering, so, there may exist θ and η in S such that $\neg better_C(\eta, \theta)$ and $\neg better_C(\theta, \eta)$. In [3, 7] several different comparators are defined. The error function $e(c\theta)$ is used. This error function returns a non-negative real number indicating how nearly constraint c is satisfied for a valuation θ . This function has the property that $e(c\theta) = 0$ if and only if $Sat(\theta, c)$. For any domain D, the trivial error function that returns 0 if the constraint is satisfied and 1 if it is not, can be used. The first of the comparators, *locally-better*, considers each constraint in C individually. The definition of this local comparator is:

Definition 1. A valuation θ is *locally-better* than another valuation η if, for each of the constraints through some level $k-1$, the error after applying θ is equal to that after applying η , and at level k the error is strictly less for at least one constraint and less than or equal for all the rest.

$$locally\text{-}better(\theta, \eta, C) \Leftrightarrow \exists k > 0 \; such \; that \; : \forall i \in 1...k-1 \;\; \forall p \in C_i \;\; e(p\theta) = e(p\eta) \wedge \exists q \in C_k \;\; e(q\theta) < e(q\eta) \wedge \forall r \in C_k \;\; e(r\theta) \leq e(r\eta).$$

Next, the *globally-better* schema for global comparators is parameterized by a function g that combines the errors of all the constraints C_i at a given level. The definition of this global comparator is:

Definition 2. A valuation θ is *globally-better* than another valuation η if, for each level through some level $k-1$, the combined errors of the constraints after applying θ is equal to that after applying η , and at level k it is strictly less.

$$globally\text{-}better(\theta, \eta, C, g) \Leftrightarrow \exists k > 0 \; such \; that \; : \forall i \in 1...k-1 \;\; g(\theta, C_i) = g(\eta, C_i) \wedge g(\theta, C_k) < g(\eta, C_k).$$

Solvers as SkyBlue or DeltaBlue use the "*locally-predicate-better*" comparator, variation of *locally-better* comparator. It may produce a very large set S of valuations. This criterion cannot compare (i.e. order) two valuations which satisfy two disjoint constraint set at a given level. A local criterion often finds solutions, that are optimal for itself, but not for the user: it is indeed too weak to discriminate which solutions

in S are really expected.To cope with this problem, we propose an incremental solver, based on a global criterion.

3 Houria System

3.1 The Criteria of Comparison used

The first criterion implemented by Houria is based on the *satisfied-count-better* comparator (1). This criterion finds solutions which satisfy the maximal number of constraints in each level of the hierarchy. This criterion uses the cardinality of the set of constraints satisfied in each level. The second criterion implemented by Houria is based on the *satisfied-count-best-case-predicate-better* comparator (2). This comparator uses the number of satisfied constraints associated with the highest label and the largest satisfaction index. The size of the set of valuations produced using this comparator is generally smaller than the one produced with the global criterion *best-case-predicate-better*. The third criterion implemented by Houria is based on the *weighted-sum-predicate-better* comparator (3). This comparator uses the sum of weights of constraints satisfied in each level of the hierarchy. The size of the set of valuations produced using one of these criteria is generally smaller than the one produced by the local criterion $locally - predicate\text{-}better$. These criteria are global and find intuitively plausible solutions at reasonable computational cost.

$$satisfied\text{-}count\text{-}better(\theta,\eta,C) \Leftrightarrow globally\text{-}better(\theta,\eta,C,g) \ where \ g(\phi,C_i) = |c : c \in C_i \wedge \neg Sat(\phi,c)|. \quad (1)$$

$$satisfied\text{-}count\text{-}best\text{-}case\text{-}predicate\text{-}better(\theta,\eta,C) \Leftrightarrow globally\text{-}better(\eta,\theta,C,g),$$
$$where \ g(\phi,C_i) \equiv (Max\ \{\overline{c}\ e(c\phi)/(c \in C_i)\}, |Max\ \{\overline{c}\ e(c\phi)/(c \in C_i)\}|)^1. \quad (2)$$

$$weighted\text{-}sum\text{-}predicate\text{-}better(\theta,\eta,C) \Leftrightarrow globally\text{-}better(\theta,\eta,C,g)$$
$$where \quad g(\rho,C_i) \equiv \left(\sum_{c\in C_i} \overline{c}\ e\ (c\rho)\right). \quad (3)$$

By (1) (resp. (2), (3)) S will not contain any solution that is worse than any other solution. S may contain multiple solutions, none of which is better than the others.

3.2 Lexicographic-Preference-Graph

In graph-theoretic terms, a constraint is considered to be satisfied if it is enforced in the solution graph. A constraint is enforced if it is included in the solution graph (i.e. the solution graph assigns a method to satisfy it). A constraint is unenforced or unsatisfied, if it is not included in the solution graph (i.e. the solution graph does not assign a method to satisfy it). A graph is admissible if it enforces all the hard constraints (i.e., the constraints in C_0). A constraint satisfier would like to choose the best of these admissible graphs. In order to obtain the best valuation that satisfies the

[1] $\overline{c}$ indicates the weight of the constraint c.

hierarchy by using one of the criteria defined in paragraph 3.1, Houria plans a lexicographic better graph, and applies the local propagation algorithm to this graph. Given one of the criteria ((1), (2) or (3)) of the *globally-better* comparator, Houria uses the constraint strengths and weights to construct a *Lexicographic-Preference-Graph* (or *LPG*) correct solution graph. A solution graph(s) is *LPG* if and only if : there are no method conflicts and no cycles, and there are no unenforced method at the k level, that if it becomes enforced then it generates a Lexicographic better solution graph. For example, given an over-constrained constraint hierarchy and the criterion (1). We suppose that each constraint contains one method. Houria may leave weaker constraints unsatisfied (unenforced in the solution graph) in order to satisfy stronger constraints. Like in Figure 1.a. This solution graph is not *LPG*, because the *strong* constraint c_2 could be enforced by selecting the method that has consequents v_4 and v_6, and revoking the *medium* constraint c_3 and the *weak* constraint c_4 , producing Figure 1.b. Actually, this solution graph is not *LPG* either, because c_5 could be enforced, producing Figure 1.c. The solution graph in Figure 1.c is *LPG* since the unenforced constraints cannot be enforced and produce a better solution graph than the solution graph in Figure 1.c.

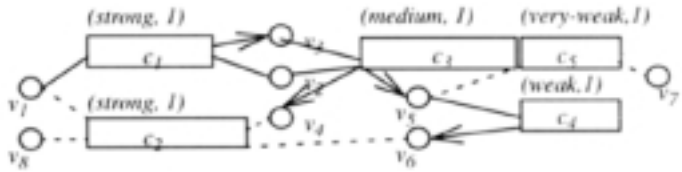

Figure 1.a - a non-LPG Solution Graph

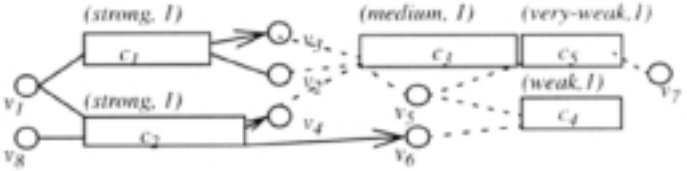

Figure 1.b - a non-LPG Solution Graph

Given an over-constrained constraint hierarchy and the criterion (2) (resp. (3)). For obtaining the *LPG* correct solution graph(s), Houria performs the following two steps : In the same class, it leaves a constraint weighted by a small satisfaction index (resp. weight) unsatisfied (i.e. unenforced in the solution graph) in order to satisfy (i.e enforce in the solution graph) the constraint weighted by a larger satisfaction index (resp. weight) in the same class, or in the other classes. Between the classes, it leaves a constraint labeled by a weaker strength and a small satisfaction index unsatisfied in order to satisfy the constraint labeled by a higher strength.

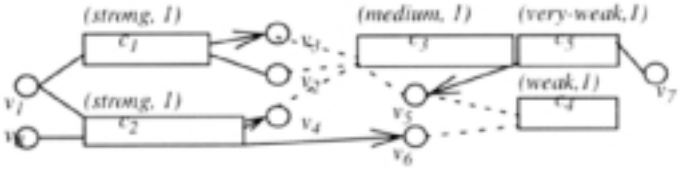

Figure 1.c - a LPG Solution Graph

3.3 General description of Houria

The set S can be seen as a set of LPG solution graphs. For obtaining a LPG solution graph, Houria[2] uses some tools. These tools are defined in paragraph 3.3.1. The algorithmic approach for computing the set S is described in the paragraph 3.3.2. In this paragraph we describe an improvement of this approach. The objective of this improvement is to reduce the number of solutions graphs developed in the set S. We describe another improvement based on the weight of the solution graphs. This improvement reduces the computation cost of the LPG.

3.3.1 Houria Tools

Notations : $\overline{c_i}$ is the pair $(label, weight)$ where $label$ is the strength of the constraint c_i and $weight$ has a several signification depending on the solution type used by the user. If the criterion (1) is used then $weight$ is equal to 1 for each constraint in the hierarchy. If the criterion (2) is used then $weight$ is the pair $(index, |index|)$ where $index$ is considered as a satisfaction index of the constraint c. If the criterion (3) is used then $weight$ is a real value associated to each constraint in the hierarchy. M_c is the set of methods of constraint c. Each m_{jc} in M_c is represented by a couple noted by $lwsg(sg, lw)$, where the sg is: $(Ant, Int, Cons)$ and lw is a list containing the pair $(label, weight)$ of the constraint c. Ant is a set containing the antecedents variables of m_{jc}. Int is a set of internal variables[3]. $Cons\ is\ a\ set$ containing the consequent variables of m_{jc}.

Example. $c = \{v_1 = v_2 - v_3\}$, $\overline{c} = (strong,\ 1)$,
$M_c = \{(v_2 - v_1 \rightarrow v_3), (v_2 - v_3 \rightarrow v_1), (v_1 + v_3 \rightarrow v_2)\}$,
$m_{1c} = ((\{v_2\ v_1\}, \{\ \}, \{v_3\}),\ (strong,\ 1)),\ m_{2c} = ((\{v_2\ v_3\}, \{\ \}, \{v_1\}), (strong,\ 1)),$
$m_3c = ((\{v_1 v_3\}, \{\ \}, \{v_2\}), (strong,\ 1))$.

A solution graph can be seen as a couple (sg, lw), where sg is the representation of the conjunction of the methods enforced in this solution graph, and lw is the list of pairs $(label, weight)$ of this solution graph. For obtaining this couple, we define the representation of the conjunction of one method and one method set by the following definition:

[2] For more details concerning Houria system, the reader, is invited to see [15, 16, 17].
[3] The set of internal variables for one method is empty.

Definition 3. let $m_{jc} = ((Ant(m_{jc}), \{\}, Cons(m_{jc})), \overline{c})$
and let $s = ((Ant(s), Int(s), Cons(s)), \overline{s})$.

$$m_{jc} \wedge s = \left(\begin{array}{l} (Ant(m_{jc}) \cup Ant(s) \backslash (Cons(s) \cup Int(s)), \\ Int(s) \cup (Ant(m_{jc}) \cap (Cons(s) \cup Int(s))) \\ \qquad \cup (Cons(m_{jc}) \cap (Cons(s) \cup Int(s))), \\ (Cons(m_{jc}) \cup Cons(s)) \backslash (Ant(s) \cup Int(s))), \\ \qquad \overline{c} \oplus \overline{s}) \end{array} \right)$$

When the criterion (1) or (3) is used, the operator $\oplus$ appends two ordered lists of pair $(label, weight)$ into one ordered list of pairs $(label, weight)$, by respectively adding the weights associated with the same label. Alternatively, when the criterion (2) is used, the operator $\oplus$ appends two ordered lists of pairs $(label, (index, |index|))$ into one, by respectively grouping the largest satisfaction indices associated with the same label.

The representation of the conjunction of one method and one method set is a couple (sg, lw). Note that the Int contains the common variables set found in both parts: the antecedent part of a method set and the consequence part of a method.

Example. $m_{1c} = ((\{v_2\ v_3\}, \{\ \}, \{v_1\}),\ (strong,\ 1))$ and $s = ((\{v_4\}, \{\ \}, \{v_2\}),$ $(medium,\ 1)),\ so\ , m_{1c}\ \wedge\ s = ((\{v_3\ v_4\}, \{v_2\}, \{v_1\}),\ ((strong,\ 1)\ (medium,\ 1))).$

The motivation for selecting this representation mode is to be able to operate on sets of variables instead of other complex data structures. We can simply compare a set of variables to know if the conjunction of one method and one method set contains conflicting methods or cycles. To attempt this goal, we define the consistency of one method and one method set.

Definition 4. let s be the representation of one method set, and C_s the representation set of connected components in s . Let m be the representation of one method.

$$\begin{array}{l} Conflict\text{-}Consistent(s, m) \Leftrightarrow ((\forall \xi \in C_s) \\ \quad Cons(\xi) \cap Cons(m) = \emptyset \wedge Int(\xi) \cap Cons(m) = \emptyset \\ \quad \wedge\ Int(m) \cap Cons(\xi) = \emptyset). \end{array}$$

$$\begin{array}{l} Cycle\text{-}Consistent(s, m) \Leftrightarrow ((\forall \xi \in C_s) \\ \quad \neg((Ant(\xi) \cap (Cons(m) \cup Int(m)) \neq \emptyset) \\ \qquad \wedge\ (Ant(m) \cap (Cons(\xi) \cup Int(\xi)) \neq \emptyset)) \\ \quad \vee\ ((\forall v_i \in (Ant(m) \cap (Cons(\xi) \cup Int(\xi))) \\ \qquad \wedge\ (\forall v_j \in (Ant(\xi) \cap (Cons(m) \cup Int(m)))) \\ \quad \Rightarrow \neg Path(v_i, v_j))). \end{array}$$

$$\begin{array}{l} Consistents(s, m) \Leftrightarrow Conflict\text{-}Consistent(s, m) \\ \qquad\qquad\qquad\qquad \wedge\ Cycle\text{-}Consistent(s, m)\ . \end{array}$$

Regarding the first condition of LPG solution graph in paragraph 3.2. The definition 4 means that : the method m is not in conflict with the method set s if and only if the predicate $Conflict\text{-}Consistent$ is true. There are no cycles between m and s if and only if the predicate $Cycle\text{-}Consistent$ is true. s is consistent with m if and only if both predicates are true (i.e. the conjunction of s and m is not consistent if and only if it contains conflicting methods or cycles).

3.3.2 The Set S Computation

S is a set of $lwsg$: (sg, lw) where sg is a solution graph and lw is the list of pairs $(label, weight)$ of this solution graph. Initially, the set S of solution graphs contains the set of solution graphs corresponding to the constraints in C_0 (i.e. the hard constraints). Houria is invoked by calling two procedures, *add-constraint* to add a constraint to each solution graph in S, and *remove-constraint* to remove a constraint from each solution graph in S. As constraints are added and removed, Houria incrementally updates each solution graph in S and sorts the set S to find the LPG solution graphs. The approach of Houria can be described by the following statements: - A constraint c is added to the hierarchical system. - If a method m of a constraint c is consistent with the solution graph in any $lwsg \in S$, then Houria forms a new $lwsg$ and adds it to S. This new $lwsg$ will have the following form: (the solution graph in $lwsg \wedge$ the method m of the constraint c , the ordered list of pairs $(label, weight)$ obtained by computing $lw \oplus (\overline{c})$).

Houria tries to add each method in the constraint c to each $lwsg$ in S. This approach is complete but costly. We propose two improvements to reduce respectively the space complexity of the set S and the computational cost of the set S. The both improvements are valid for any global predicate comparator.

Improvement for reducing the space complexity of S

The first improvement to this approach consists in keeping in S only the maximal elements, *i.e.* when Houria tries to add m to the $lwsg$, (m is one method of constraint c , $lwsg$ is one couple in S) we can make the following distinctions:

When m is consistent with the solution graph in $lwsg$, Houria updates $lwsg$ by adding m to the solution graph in $lwsg$, and adding ($\overline{c}$) to lw by using the operator $\oplus$.

When m is not consistent with the solution graph in $lwsg$, Houria updates S by executing the following steps:

- Extract from the solution graph in $lwsg$ a set of $lwsg$ (noted S'), where m is consistent with the solution graph in each $lwsg$ in S'.

- For any $lwsg \in S'$, the solver forms a new $lwsg$ by considering the method m.

- Add this new $lwsg$ to S if it is maximal (*i.e.* if it does not exist another $lwsg$ in S, which contains this new $lwsg$).

This improvement reduces the size of S, while still preserving the completeness of the solver.

Improvement for reducing the computational cost of S

Another improvement to this approach aims at reducing the cost of the solver, this enhancement used the list of pairs $(label, weight)$ of all $lwsg$ in S.

S is partitioned into subsets, where each subset contains the $lwsg$ that have an equal list of pairs $(label, weight)$. Each subset is associated with a queue called queue of constraints to add. Each subset has two weights: *current-label-weight* and *potential-label-weight*. The *current-label-weight* of a subset is equal to the lw of one $lwsg$ in this subset. The *potential-label-weight* of a subset is equal to an ordered list of pairs $(label, weight)$. This ordered list is obtained by applying the operator $\oplus$ on the *current-label-weight* and all pairs $(label, weight)$ of the constraints in the associated queue of this subset. All subsets in S will be lexicographically ordered on respect to their *potential-label-weight*. The *current-label-weight* and *potential-label-weight* with the first subset in S are always equal. When a constraint c is added to the system, Houria tries to add it to each $lwsg$ in the first subset of S, and keeps this constraint c in all queues associated to the other subsets of S. Houria halts when the *current-label-weight* of the first subset in S is not lower than the *potential-label-weight* of the second subset in S. Otherwise, Houria tries to add all constraints in the queue of the second subset to any $lwsg$ in this second subset. The result subset of this tentative is ordered and placed in S.

The goal of this improvement is to delay the computations as much as possible, and to not enumerate all the LPG solution graphs. This means that Houria performs the computations only if we are certain that a better solution graph may exist.

Another heuristic is considered by this approach. This heuristic consists to perform on priority the subset which has the highest *current-label-weight* from all subsets which have the same *potential-label-weight*. In average this improvement reduces the complexity of computing the LPG solution graphs while still being complete.

3.3.3 Removing Constraints

In order to remove a constraint c, Houria partitions the set S into two sets S' and S''. S'' contains the subsets which associated queues include the constraint c to remove. The procedure removes the constraint c by substracting this constraint from any queue of the subset in S''. S' contains some subsets where an $lwsg$ in each subset contains one method of the constraint c to be removed. The procedure *Remove-constraint* removes all methods of the constraint c from all $lwsg$ in each subset in S' , and keeps each $lwsg$ in each subset of S' maximal consistent. The two sets S' and S'' are sorted

by considering the *potential-label-weight* of their subsets. The procedure returns the first subset of S, this first subset contains the LPG solution graph.

3.4 Implementation and Measurements

In the worst case, Houria is exponential in time since it is based on the best-first search strategy and it handles a global criterion. But in practice, the observed running time of Houria is actually acceptable giving an upper-bound number of constraints.

Houria is implemented in Lisp. In order to assess out the performance of our solver, we generate a random constraint system based on two parameters: the constraint number cn and the constraint arity ca.

We have run two sets of experiments, the first with ca=2 (Figure 2.a) the second with ca=3 (Figure 2.b). With ca=2 the number of generated methods is 2 and with ca=3, the the number of generated methods is randomly chosen in {2..6}. For each set generated, the number of constraints range from 10 to 100 by steps of 10. In each test, the number of variables is the third (i.e. 1/3) of the number of constraints generated each steps.

The performance reported for Houria includes the amount of time required to find the LPG solution graphs[4]. We have repeated the test for 50 different random over-constrained problems, reporting the average of the results. The graphs (in seconds) in Figures 2.a and 2.b can be interpreted as follows : with ca =2 or ca =3, the solver time to plan a LPG solution graph is acceptable. When the size of the problem is too large(> 100), the solver time to plan a LPG solution graph is long.

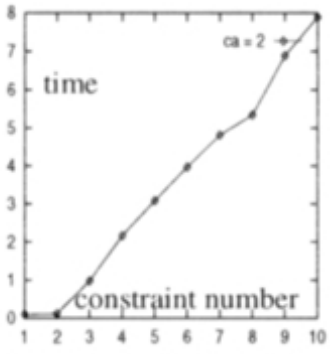

Figure 2.a - planning time $ca = 2$

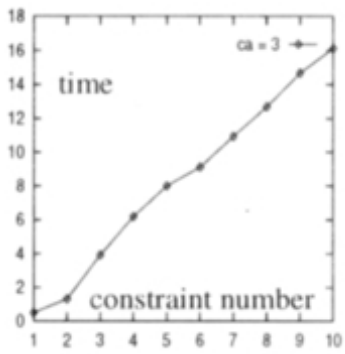

Figure 2.b - planning time $ca = 3$

[4] The empirical test for planning the LPG by using the criterion (2).

4 The Use of Houria in CLP Languages

As mentioned by Wilson in [13] experience with writing programs in $HCLP(R, locally\text{-}better)$ has provided many examples where the local comparator may rule out non-intuitive solutions. This results from the restriction of the comparator to select among valuations arising from a single constraint hierarchy.

4.1 Extended Theory of Constraint Hierarchies

Wilson and Borning extend in [13] the constraint hierarchy theory to multiple constraint hierarchies. This extension consists of defining the set of solutions to many constraint hierarchies. This lays the theoretical foundation for inter-hierarchy comparators and will allow to rule out the non-intuitive solutions and to eliminate the undesirable solutions. A solution to a set of constraint hierarchies H will consist of a valuation for all free variables in H. Normally, the set H will consist of hierarchies that arise from alternate rule choices in a program and the set S of solutions contains all solutions to H , rather than just to a single hierarchy. Since the *locally-better* comparators consider each constraint in the hierarchy individually to compare how well different valuations satisfy that constraint, they are not redefined to compare solutions between different hierarchies. In other words, *locally-better* is defined only if H consists of a single hierarchy. Because the *globally-better* comparators take some aggregate measure to combine the errors obtained in each level of the hierarchy, they are extended to compare valuations arising from different hierarchies.

Definition 5. A valuation θ_h is *globally-better* than another valuation $\eta_{h'}$ if, for each level through some level $k-1$, the combined error g of the constraints after applying θ to the constraints in hierarchy $h \in H$ is equal to that after applying η to the constraints in hierarchy $h' \in H$ and at level k it is strictly less. (h and h' can be either the same or different hierarchies. If they are the same, then the following definition is equivalent to the one for intra-hierarchy comparison.)

4.2 Algorithm for Inter-hierarchy Comparisons

Based on this extension of comparison and on Houria system, we present an efficient algorithm for comparisons between the hierarchies that arise from alternate rule choices in a program. This algorithm consists on two procedures (*Initialization* and *Best-Desirable-Solution*) and is based on the Houria system since Houria system, which, based on global criteria, allows inter-hierarchy comparisons.

The first step in the *Initialization* procedure is to form the set $H = \{h_1, h_2, ...h_n\}$ of the hierarchies resulting from the alternate rule choices in a program, after goals have been successfully reduced. This first step can be achieved as in CLP, temporarily ignoring the soft constraints, except to accumulate them and form the set H. The procedure initializes the variable *Comp* with the criterion of the global comparator used in the program ((1) (2) or (3)). Subsequently, the procedure examines the set H of hierarchies and associates to each hierarchy a variable $V\text{-}free$ that contains

the set of the free variables in the hierarchy (i.e. the set of variables, that have not been yet determined by the first step). A variable *CN* is associated with each hierarchy in H and it contains the set of the constraints of this hierarchy. A list of the eligible constraints, denoted *CN-eligible*, is associated with each hierarchy in H. The procedure examines each constraint c in h. If the set of variables constrained by c contains at least one free variable (in $V\text{-}free$) then the constraint c is added to the list of the eligible constraints. This list will be used in order to determine the values of the free variables. The procedure forms another list noted by *CN-check* for each hierarchy. This list contains the set of constraints, that do not constrain a free variable (i.e. all the variables constrained are bound). After that, since the objective of our approach is to find the "preferred" hierarchy (i.e the hierarchy that produces the best valuation to satisfy the soft constraints), we must consider the strength and the weight of the constraints satisfied in the list *CN-check*. In fact, this operation is achieved by the procedure. The procedure associates to each hierarchy in H a variable noted by *initial-label-weight* initialized by the list of pairs $(label, weight)$ of the constraints satisfied in the list *CN-check*. Some of the eligible constraints in the list *CN-eligible* may constrain bound variables (i.e. not free variables). These bound variables are marked by the read-only annotation[5], because they have been determined by the hard constraints (the predicates in the program) in the first step of the procedure.

In the *Best-Desirable-Solution* procedure, we first eliminate from H the set of the hierarchies where the sum of the *initial-label-weight* and the *label-weight*(*CN-eligible*) of each hierarchy in this set is strictly less than the *initial-label-weight* of another hierarchy in H (since we need that the free variables must be computed by the strongest hierarchy (i.e. that have the maximum *label-weight*)). The resulting set H after this operation is ordered by the criterion sum of the *initial-label-weight* and the *label-weight*(*CN-eligible*) decreasing. For a hierarchy in the ordered set H, the procedure calls Houria solver in order to determine the maximum subset of the constraints in the *CN-eligible* (of this hierarchy) that can be solved and produce the best valuation by respecting the criterion used. The consequence of this call is the assignment of the variables Gr and *label-weight-Gr* that contain, respectively, the best graph of the methods of the constraints in the list *CN-eligible*, and the list of pairs $(label, weight)$ of this graph. This operation is executed until the set H is empty or until the sum of the *initial-label-weight* and the *label-weight-Gr* of the graph resulting from the last call for a hierarchy in H is lexico-graphically greater or equal than the sum of the *initial-label-weight* and *label-weight*(*CN-eligible*) of the next hierarchy in H (i.e. if this condition is satisfied then it is not possible to find a hierarchy from the rest of the hierarchies in H that gives a better solution since H is ordered). This second case is an improvement which reduces the number of calls to Houria, while still preserving the completeness of the algorithm. All the hierarchies performed in the precedent operation are stored in H'. The procedure keeps in H' only the set of hierarchies where the sum of the *label-weight-Gr* and the *initial-label-weight* of each hierarchy in this set is maximal. The procedure calls Houria solver to solve each hierarchy in H'. This resolution consists of the execution of the methods in the graph and the free

[5] The read-only annotation has been introduced in [5].

variables are computed. The set of variables of the hierarchy is returned containing the desirable answer of the program.

In theory the extended definition allows inter-hierarchy comparisons. The algorithm proposed in this section is based on these extended definitions and on the versions of the Houria solver. This algorithm can be incorporated in the CLP languages to allow the ability to execute inter-hierarchy comparisons. Thus, we can obtain the elimination of the undesirable solutions in many applications that contain constraint hierarchies. This algorithm looks particularly promising when the number of the alternate rule choices of each predicate in the program is not very large since the more the solution is to the right in the tree search, the better this procedure is. In the opposite, Houria can be used in a simple Branch and Bound algorithm since the more the solution is to the left in the tree search, the more this procedure converges to the Branch and Bound. For more details concerning this approach, the reader is invited to see[18].

4.3 Example of a HCLP Program

The following is an example of a $HCLP$ program that illustrates the work of the two procedures in the previous section. The comparator used in this program is the $satisfied\text{-}count\text{-}better(1)$.

$F(x,y,z)$:- $G(x)$, $(strong,\ 1)\ \ x = 2y + z$, $(medium,\ 1)\ \ x = 2y$, $(weak,\ 1)\ \ x > 4$.

$F(x,y,z)$:- $G(x)$, $(strong,\ 1)\ \ x > 8$, $(medium,\ 1)\ \ x = 2y + z$, $(weak,\ 1)\ \ x = 2y$.

$F(x,y,z)$:- $K(x,y)$, $(strong,\ 1)\ \ x = 2y$, $(medium,\ 1)\ \ x = 2y+z$, $(medium,\ 1)\ \ y = 2x + z$, $(very\text{-}weak,\ 1)\ \ x > 4$.

$G(4)$.

$G(8)$.

$K(1,3)$.

$K(6,3)$.

Given the goal $F(A,B,H)$, the first step of the $Initialization$ procedure would return the set H that contains six hierarchies $(h_1..h_6)$ resulting from the alternate rule choices of the predicates G and K.

$h_1 = \{\ required\ \ A = 4$, $(strong,\ 1)\ \ A = 2B+C$, $(medium,\ 1)\ \ A = 2B$, $(weak,\ 1)\ \ A > 4\}$

$h_2 = \{\ required\ \ A = 8$, $(strong,\ 1)\ \ A = 2B+C$, $(medium,\ 1)\ \ A = 2B$, $(weak,\ 1)\ \ A > 4\}$

$h_3 = \{ \textit{required}\ A = 4,\ (\textit{strong},\ 1)\ A > 8,\ (\textit{medium},\ 1)\ A = 2B+C,\ (\textit{weak},\ 1)\ A = 2B\}$

$h_4 = \{ \textit{required}\ A = 8,\ (\textit{strong},\ 1)\ A > 8,\ (\textit{medium},\ 1)\ A = 2B+C,\ (\textit{weak},\ 1)\ A = 2B\}$

$h_5 = \{ \textit{required}\ A = 1,\ \textit{required}\ B = 3,\ (\textit{strong},\ 1)\ A = 2B,\ (\textit{medium},\ 1)\ A = 2B + C,\ (\textit{medium},\ 1)\ B = 2A + C,\ (\textit{very-weak},\ 1)\ A > 4\}$

$h_6 = \{ \textit{required}\ A = 6,\ \textit{required}\ B = 3,\ (\textit{strong},\ 1)\ A = 2B,\ (\textit{medium},\ 1)\ A = 2B + C,\ (\textit{medium},\ 1)\ B = 2A + C,\ (\textit{very-weak},\ 1)\ A > 4\}$.

The result of the *initialization* procedure on the hierarchies in the set H is the following:

$(V\text{-}free_{h1},\ CN\text{-}eligible_{h1},\ CN\text{-}check_{h1},\ initial\text{-}label\text{-}weight_{h1}) = (\{B, C\},$
$\{(\textit{strong},\ 1)\ A = 2B + C,\ (\textit{medium},\ 1)\ A = 2B\},\ \{(\textit{weak},\ 1)\ A > 4\}, \{\})$.

$(V\text{-}free_{h2},\ CN\text{-}eligible_{h2},\ CN\text{-}check_{h2},\ initial\text{-}label\text{-}weight_{h2}) = (\{B, C\},$
$\{(\textit{strong},\ 1)\ A = 2B+C,\ (\textit{medium},\ 1)\ A = 2B\},\ \{(\textit{weak},\ 1)\ A > 4\},\ \{(\textit{weak},\ 1)\})$.

$(V\text{-}free_{h3},\ CN\text{-}eligible_{h3},\ CN\text{-}check_{h3},\ initial\text{-}label\text{-}weight_{h3}) = (\{B, C\},$
$\{(\textit{medium},\ 1)\ A = 2B + C,\ (\textit{weak},\ 1)\ A = 2B\},\ \{(\textit{strong},\ 1)\ A > 8\}, \{\})$.

$(V\text{-}free_{h4},\ CN\text{-}eligible_{h4},\ CN\text{-}check_{h4},\ initial\text{-}label\text{-}weight_{h4}) = (\{B, C\},$
$\{(\textit{medium},\ 1)\ A = 2B + C,\ (\textit{weak},\ 1)\ A = 2B\},\ \{(\textit{strong},\ 1)\ A > 8\}, \{\})$.

$(V\text{-}free_{h5},\ CN\text{-}eligible_{h5},\ CN\text{-}check_{h5},\ initial\text{-}label\text{-}weight_{h5}) = (\{C\},$
$\{(\textit{medium},\ 1)\ A = 2B + C,\ (\textit{medium},\ 1)\ B = 2A + C\},\ \{(\textit{strong},\ 1)\ A = 2B,\ (\textit{very-weak},\ 1)\ A > 4\}, \{\})$.

$(V\text{-}free_{h6},\ CN\text{-}eligible_{h6},\ CN\text{-}check_{h6},\ initial\text{-}label\text{-}weight_{h6}) = (\{C\},$
$\{(\textit{medium},\ 1)\ A = 2B + C,\ (\textit{medium},\ 1)\ B = 2A + C\},\ \{(\textit{strong},\ 1)\ A = 2B,\ (\textit{very-weak},\ 1)\ A > 4\},\ \{(\textit{strong},\ 1),\ (\textit{very-weak},\ 1)\})$.

Since $initial\text{-}label\text{-}weight_{h6} >_{lex} (weight(CN\text{-}eligible_{h5}) \oplus initial\text{-}label\text{-}weight_{h5})$ (i.e. $((\textit{strong},\ 1),\ (\textit{very-weak},\ 1)) >_{lex} ((\textit{medium},\ 1),\ (\textit{medium},\ 1))$) the *Best-Desirable-Solution procedure* eliminates the hierarchy h_5 from H. Also, for the same reasons the hierarchies h_3 and h_4 are eliminated from H. The resulting set H is ordered by the criterion decreasing sum of the $initial\text{-}label\text{-}weight_{hi}$ and the $label\text{-}weight$ $(CN\text{-}eligible_{hi})$, and contains now : $\{h_6, h_2, h_1\}$. The procedure calls the Houria solver in order to perform the eligible constraints in the hierarchy h_6. For the constraints in $CN\text{-}eligible_{h6}$, Houria returns the graph Gr_{h6} that contains the only method $C \leftarrow A - 2B$ (i.e. the value of the variable C can be computed by considering the value of A and of B) (the other methods are not used because the

variables A and B are marked with the read-only annotation). The content of *label-weight-Gr*$_{h6}$ returned by Houria is ((*medium*, 1)). (The other alternate graph that contains the method $C \leftarrow B - 2A$ can be returned by Houria, but since the *label-weight* of both graphs is the same, we only give one solution). The hierarchy h_6 is now eliminated from H and stored in H'. Since the weight of the resulting graph is less than the total weight of the eligible constraints of the hierarchy h_6 (i.e. *label-weight-Gr*$_{h6}$ $<_{lex}$ *label-weight*(*CN-eligible*$_{h6}$) because ((*medium*, 1)) $<_{lex}$ ((*medium*, 1), (*medium*, 1))), then we are not certain that the optimal solution has been obtained. The procedure performs the previous operation on the hierarchy h_2. The result from the Houria calls is : GR_{h2} that contains the methods: $C \leftarrow A - 2B$ and $B \leftarrow A \backslash 2$. The content of *label-weight-Gr*$_{h2}$ returned by Houria is ((*strong*, 1), (*medium*, 1)). The procedure halts to perform the rest of the hierarchies in H, since the set H is ordered and the sum of *label-weight-Gr*$_{h2}$ and *initial-label-weight*$_{h2}$ is lexicographically greater than the sum of *label-weight*(*CN-eligible*$_{h1}$) and *initial-label-weight*$_{h1}$ (i.e. we are certain that one of the "*best*" hierarchies has been performed). The set H' contains now both of the hierarchies h_6 and h_2. The hierarchy h_6 is eliminated from the set H' because the sum of *label-weight-Gr*$_{h6}$ and the *initial-label-weight*$_{h6}$ is less than the sum of *label-weight-Gr*$_{h2}$ and the *initial-label-weight*$_{h2}$ (i.e. ((*strong*, 1), (*medium*, 1), (*very-weak*, 1) $<_{lex}$ ((*strong*, 1), (*medium*, 1), (*weak*, 1))). The procedure resolves GR_{h2} by applying the procedure in Houria which solves this graph and the desirable solution $A = 8$, $B = 4$, $C = 0$, is returned.

5 Conclusions

Houria is a solver which incrementally handles a constraint hierarchy where each class in the hierarchy can contain constraints weighted with different weights. It is particulary suitable in applications where the user desires to have more than one solution, e.g. applications such as graphic design environments [14], where the user wants to debug a constraint network, and explore alternative solutions. Houria can be a good solution to commonly-encountered constraint problems in graphical layout and visual languages. Also, Houria can be used via the algorithm proposed in section 4. It allows inter-hierarchy comparisons then we obtain the elimination of the undesirable solutions in many applications in CLP languages that contain functional constraint hierarchies, such as geometric layout [2], physical simulations, and user interface design [1], document formatting, algorithm animation, and design and analysis of mechanical devices and electrical circuits.

6 Future Work

Houria will be extended to support also a non-functional constraints (i.e. numerical constraints, interval constraints, etc.). Houria will develop solutions for addressing a large sub-set of over-constrained problems and mixed-initiative resolution of complex, constrained optimisation tasks.

Acknowledgement

I thank the anonymous referees for their comments on a preliminary version of this paper. I am grateful to Pr. Abdelhamid Benchakroun, Dr. Geir Hasle and to Dr. Taoufik Bouzoubaa, for their help and support.

References

[1] M. Sannella & A. Borning, *Multi-Garnet: Integrating multi-way constraints with garnet*, Tech. Rep. 92-07-01, Departement of Computer Science and Engineering, University of Washington, Sep. 1992.

[2] A. B. Myers, D. Giuse, R. B. Dannenberg, B. Vander Zanden, D. Kosbie, P. Marchal & E. Pervin, "Comprehensive support for graphical, Highly-Interactive User Interfaces: The Garnet User Interface Development Environment". *IEEE Computer*, 23(11):71-85, Nov. 1990.

[3] A. Borning, S. Maher, M. A. Martindale, & M. Wilson, Constraint hierarchies and logic programing". In *Proceedings of the Sixth International Logic Programing Conference*, pp 149-164, 1989.

[4] M. Sannella, *The SkyBlue Constraint Solver*. Tech. Rep. 92-07-02, Dep. of Comp. Sc. and Eng. , University of Washington, Feb. 1993.

[5] A. Borning, B. Freeman-Benson, & M. Wilson, "Constraint hierarchies". *Lisp and Symbolic Computation*, Vol. 5, pp. 221-268, 1992.

[6] A. Borning & M. Wilson, *Hierarchical Constraint Logic Programing*. Tech. Rep. 93-01-02a, Departement of Computer Science and Engineering, University of Washington, May. 1993.

[7] J. H. Maloney, A. Borning & B. N. Freeman-Benson, "Comstraint technology for user-interface construction in ThinglabII". In: *Proceedings of the ACM Conference on Objet-Oriented-Programing Systems Languages and Applications*, pp 381-388, Oct . 1989.

[8] M. Sannella, B. Freeman-Benson, J. Maloney & A. Borning, *Multi-way versus One-Way Constraints in User Interfaces: Experience With The Deltablue solver*. Tech. Rep. 92-07-05, Departement of Computer Science and Engineering, University of Washington, July. 1992.

[9] B. A. Myers, D. A. Giuse, R. B. Dannenberg, B. Vander Zanden, D. S. Kosbie, E. Pervin, A. Mickish & P. Marchal, "Garnet: Comprehensive Support for Graphical, Highly Interactive User Interfaces". *IEEE Computer*, vol. 23, no. 11, pp. 71-85, Nov. 1990.

[10] A. Borning, "The Programming Languages Aspects of Thinglab, a Constraint-Oriented Simulation Laboratory". *ACM Transactions on Programming Languages and Systems*, vol. 3, no. 4, pp. 353-387, Oct. 1981.

[11] H. Hosobe, K. Miyachit, S. Takahash, S. Matsuoka, A. Y, *Locally Simultaneous Constraint Satisfaction* LNCS 874: PPCP, Nov. 1994.

[12] G. J. Sussman, & G.L., Steele, "CONSTRAINTS-S langage for expressing almost-hierarchical descriptions". *A.I. 14*, 1, pp 1-39, Jan. 80.

[13] A. Borning & M. Wilson, "Extending Hierarchical Constraint Logic Programming: Nonmonotonicity and Inter-Hierarchy Comparison". In: *Proceedings of the North American Conf. on LP*, Cleveland, Oct. 1989.

[14] J. H. Maloney, A. Borning & B. N. Freeman-Benson, "Comstraint technology for user-interface construction in thinglabII". In: *Proceedings of the ACM Conference on Objet-Oriented-Programing Systems Languages and Applications*, pp 381-388, Oct . 1989.

[15] M. Bouzoubaa, B. Neveu, G. Hasle, "Computer Science and Operations Research: Recent Advences in The Interface", chapter *Houria III: Planning of Lexicographic Weight Sum Better Graph for Equational Constraints.* INFORMS, CSTS, Dallas, Texas, January 1996.

[16] M. Bouzoubaa, "The Houria constraint solver". In: *Proc. of the International Conference on Applications of Artificial Intelligence in Engineering X(AIEng'95)*, Udine, Italy, July 1995.

[17] M. Bouzoubaa, B. Neveu, G. Hasle, "Houria II: A solver for hierarchical systems, planning of lexicographic satisfied count best case better graph for equational constraints". In: *Constraint for Graphics and Visualization CP-95*, Cassis, France, September 1995.

[18] M. Bouzoubaa, "Functional Constraint Hierarchies". In: *Proc. of the Principles and Practice of Constraint Programming*, CP-96, LNCS 1118, August 1996.