Inter-Hierarchy Comparison in HCLP[1]

Roman Barták

Department of Theoretical Computer Science
Charles University
Malostranské náměstí 2/25
Praha 1, Czech Republic

e-mail: `bartak@kti.mff.cuni.cz`
URL: `http://kti.ms.mff.cuni.cz/~bartak/`
phone: +420-2 2191 4242
fax: +420-2 2191 4323

Abstract

Constraint hierarchies have been proposed to describe over-constraint systems of constraints by specifying constraints with hierarchical preferences, i.e., hard and soft constraints. While the hard (required) constraints must hold, the soft (preferential) constraints should be satisfied as much as possible depending on the criterion used. Currently, constraint hierarchies are mostly applied to the areas of graphical user interfaces and geometric layout, but the area of usage is much more wider.

The integration of constraint hierarchies with Constraint Logic Programming (CLP) is called Hierarchical Constraint Logic Programming (HCLP). In the original definition of HCLP, only alternate solutions to one constraint hierarchy are compared and the best solutions are returned. The later extension of HCLP also enables comparison of solutions to more constraint hierarchies arising from different choices of rules in HCLP program. This extension, called inter-hierarchy comparison, extends usefulness of HCLP programs by eliminating non-intuitive solutions. However, at the same time, it introduces nonmonotonic behaviour in HCLP programs which creates novel implementation problems.

In this paper we present an algorithm for efficient solving of constraint hierarchies using inter-hierarchy comparison within HCLP. The efficiency of the proposed algorithm is based on tight integration with the HCLP interpreter and on elimination of useless branches of computation. As the algorithm makes an extension of hierarchical constraint solvers which support global comparators, its efficiency can be further improved by harmonizing with underlying hierarchical constraint solver. We present such an extension that uses properties of our generalized approach to solving constraint hierarchies.

Keywords: constraint hierarchies, HCLP, inter-hierarchy comparison

---

[1] The research was partially supported by the Grant Agency of Czech Republic under the contract No 201/96/0197.

1 Introduction

Constraint hierarchies were introduced for describing over-constrained systems of constraints by specifying constraints with hierarchical strengths or preferences[2]. It allows one to specify declaratively not only the constraints that are required to hold, but also weaker, so called soft constraints at an arbitrary but finite number of strengths. Weakening the strength of constraints helps to find a solution of previously over-constrained system of constraints. Intuitively, the stronger a constraint is, the more it influences the solution of the hierarchy. Moreover, constraint hierarchies allow "relaxing" of constraints with the same strength by applying, e.g., weighted-sum, least-squares or similar methods.

This constraint hierarchy scheme can be parameterized by a comparator C that allows one to compare different possible solutions to a single hierarchy and to select the best ones. Currently, there are two widely used groups of comparators, namely locally-better and globally-better comparators. While the locally-better comparators consider each constraint individually, the globally-better comparators combine errors of all constraints at a given level using some combining function. Thus, the globally-better comparators can be used for inter-hierarchy comparison [13], i.e., comparison of solutions to two or more constraint hierarchies.

The constraint hierarchies are used in Hierarchical Constraint Logic Programming (HCLP) which is an extension of Constraint Logic Programming (CLP). In the original definiton of HCLP [4,14] only alternate solutions to one constraint hierarchy are compared and the best solutions are returned. The later extension of HCLP [13] also enables comparison of solutions to more constraint hierarchies arising from different choices of rules in HCLP program. Such comparison, called inter-hierarchy comparison (opposite to more common intra-hierarchy comparison), rules out non-intuitive solutions and, in our opinion, it is closer to declarative character of HCLP programs. Then, the HCLP programs with inter-hierarchy comparison can be used as rule-based expert systems [2]. At the same time, the inter-hierarchy comparison introduce nonmonotonic behaviour in HCLP programs that creates novel implementation problems.

In this paper we present an efficient algorithm for solving constraint hierarchies using inter-hierarchy comparison. This algorithm is intended to be used within the HCLP framework and, thus, it can take advantage of tight integration with the HCLP interpreter. Opposite to CLP/HCLP approach that does not change the operational behaviour of logic programs, i.e., usage of depth-first search method, the proposed algorithm utilizes breadth-first search approach to solving HCLP programs. HCLP interpreter supporting inter-hierarchy comparison has to explore the whole computation tree of the program to collect all possible constraint hierarchies arising from different choices of rules in HCLP program. We propose to use breadth-first search to explore the computation tree combining with the evaluation of partial solutions. This enables us to eliminate useless branches (subtrees) of the computation tree and, thus, to dramatically improve efficiency of the algorithm. This approach is more efficient than other approaches which have to collect all constraint hierarchies first. Additionaly, the proposed algorithm makes an extension of hierarchical constraint solvers which support global comparators. Therefore, the efficiency of the algorithm can be further improved by harmonizing with the underlying hierarchical constraint solver.

The paper is organized as follows. In Section 2, we give a background of our research and we outline the theory of constraint hierarchies and inter-hierarchy comparison. In Section 3, we give a practical motivation for studying hierarchical constraint solvers supporting inter-hierarchy comparison. In Section 4, we outline the methodology and theoretical foundation of our approach. In Section 5, we describe the

---

[2] Another method for describing over-constrained systems is PCSP (Partial Constraint Satisfaction Problems).

skeleton of proposed algorithm for solving constraint hierarchies using inter-hierarchy comparison. We concentrate on its integration with HCLP interpreter there. In Section 6, we give an example that ilustrates the work of the proposed algorithm. Section 7 is dedicated to further improvements of the proposed algorithm via harmonizing with underlying hierarchical constraint solver. We sketch our generalized approach to solving constraint hierarchies there and then we describe how one may utilize its properties to improve efficiency of inter-hierarchy comparison. We conclude with some final remarks and description of future research.

2 Constraint hierarchies and inter-hierarchy comparison

The theory of constraint hierarchies was developed in [4]. It is based on the idea of labeling each constraint by preference that determines the strength of the constraint. The constraint hierarchy is then a finite set of labeled constraints. Intuitively, the stronger a constraint is, the more it influences the solution of the hierarchy. The solution of the constraint hierarchy H can be formally expressed in a following way:

$$S_0 = \{ \theta \mid \forall c \in H_0 \ \ c\theta \text{ holds} \}$$

$$S = \{ \theta \mid \theta \in S_0 \ \& \ \forall \sigma \in S_0 \ \neg \ \text{better}(\sigma,\theta,H) \},$$

where θ, σ are valuations of free variables in constraints from H, H_0 is a vector of required constraints in H (in some arbitrary order, with their labels removed) and *better* is a relation comparing two valuations according to the hierarchy H. The set S_0 contains valuations satisfying required constraints and S is a solution set.

There is a number of reasonable candidates for the predicate *better* which is called a *comparator* and it is a parameter to the constraint hierarchy scheme. Currently, two groups[3] of comparators are used: locally-better and globally-better comparators.

The *locally-better* comparators consider each constraint individually. They are defined by the following way:

$$\textit{locally-better}(\theta,\sigma,H) \equiv_{\text{def}}$$
$$\exists k>0 \ \forall i \in \{1,\dots,k-1\} \ \forall c \in H_i \ \ e(c,\theta)=e(c,\sigma) \ \&$$
$$\exists c' \in H_k \ e(c',\theta)<e(c',\sigma) \ \& \ \forall c \in H_k \ e(c,\theta) \le e(c,\sigma),$$

where $e(c,\theta)$ is an *error function* that returns a non-negative real number indicating how nearly a constraint c is satisfied for a valuation θ and H_i are levels of the hierarchy H defined in an obvious way, i.e., H_1 is a vector of strongest non-required constraints in H etc.

The *globally-better* comparators combine errors of all the constraints at given level H_i using a combining function g, and then compare the combined errors. They are defined as follows:

$$\textit{globally-better}(\theta,\sigma,H,g) \equiv_{\text{def}}$$
$$\exists k>0 \ \forall i \in \{1,\dots,k-1\} \ g(\theta,H_i)=g(\sigma,H_i) \ \&$$
$$g(\theta,H_k)<g(\sigma,H_k).$$

Note that the scheme of globally-better comparator is parameterized by the combining function g that combines errors of individual constraints via, e.g., weighted sum, worst case or least squares methods.

3 Recent definition of constraint hierarchies [14] also supports another type of comparator called regionally-better.

Constraint hierarchies extend the Constraint Logic Programming CLP(R) scheme parameterized by the domain R of constraints into a more general scheme Hierarchical Constraint Logic Programming HCLP(R,C) parameterized by the domain R and the comparator C. These languages provide both required constraints and default constraints of various strengths. The following program [13] is an example of HCLP program.

```
f(X):-strong X>3, g(X).
g(5).
g(1).
```

Given the goal `?-f(A)`, HCLP would first return the answer `A=5,` and on backtracking `A=1`. In reality, there are produced two constraint hierarchies arising from different choices of the rule `g`. One is hierarchy `required A=5,strong A>3` and the other is `required A=1,strong A>3`. The above definition of constraint hierarchies does not prefer neither the first nor the second answer as the definition allows intra-hierarchy comparison only.

To rule out unintuitive solutions described above, the extended definition of constraint hierarchies with inter-hierarchy comparison was proposed in [13]. It is a definition of the solution set for the set of constraint hierarchies Δ. Note that each valuation in sets S_0, S is labeled by its respective constraint hierarchy.

$$S_0 = \{ \theta_H \mid \forall c \in H_0 \;\; c\theta \text{ holds} \}$$

$$S = \{ \theta_H \mid \theta_H \in S_0 \;\&\; \forall \sigma_J \in S_0 \; \neg \; \text{better}(\sigma_J,\theta_H,\Delta) \}.$$

Because the locally-better comparators consider each constraint in the hierarchy individually and the comparator *better* compares valuations arising from more hierarchies now, the locally-better comparators are not usually redefined to compare solutions to different hierarchies[4]. However, the definiton of globally-better scheme can be naturally extended to compare valuations arising from different hierarchies.

$$\textit{globally-better}(\theta_H,\sigma_J,\Delta,g) \equiv_{\text{def}}$$
$$\exists k>0 \; \forall i \in \{1,\ldots,k-1\} \;\; g(\theta_H,H_i)=g(\sigma_J,J_i) \;\; \&$$
$$g(\theta_H,H_k)<g(\sigma_J,J_k).$$

Again, the globally-better scheme is parameterized by some combining function *g*. Note that the combining function *g* is applied to the valuation and its respective constraint hierarchy.

3 Motivation

Although the concept of inter-hierarchy comparison is not new at all [13], it is used scarce in practical applications if at all. Even the original constraint hierarchies with intra-hierarchy comparison are used sparingly, mostly in areas of graphical user interfaces and geometric layout. It sounds surprisingly because of the nice declarative description of over-constrained systems by constraint hierarchies.

The problem is probably hidden in hierarchical constraint solvers which utilize different concepts than traditional constraint satisfaction. Additionaly, most current hierarchical constraint solvers are constructed for locally-better comparators only [8,9,11,12], and thus they are not suitable for inter-hierarchy comparison. Also, if the constraint solvers with intra-hierarchy comparison are computational-hungry, the solvers supporting inter-hierarchy comparison are even more demanding for computational power.

[4] There exists definition of constraint hierarchies which allows locally-better comparators in inter-hierarchy comparison.

As we do not see much demand for inter-hierarchy comparison in current applications utilizing constraints, we turn our attention to new areas. In [2] we proposed to use HCLP with inter-hierarchy comparsion to write rule-based expert systems where constraint hierarchies can naturally handle uncertaint information. In such systems, the existence of algorithms supporting inter-hierarchy comparison becomes crucial.

4 Theoretical basics

Original HCLP systems with intra-hierarchy comparison does not change the operational behaviour of CLP or Prolog respectively, i.e., they use the depth-first search to explore the computation tree. Consequently, the HCLP system can be built over the underlying CLP system. The approach of HCLP with inter-hierarchy comparison is a bit different as it requires to collect all possible constraint hierarchies arising from different choices of rules in HCLP program.

The algorithm for inter-hierarchy comparison presented in [6] uses the underlying HCLP system to collect constraint hierarchies and then it solves the resulting set of constraint hierarchies. The advantage of this approach is that it does not require to change the operational behaviour of HCLP, i.e., the depth-first search. Furthermore, this algorithm is not confined to HCLP framework, i.e., to the way of collecting constraint hierarchies, and thus it could be used in all applications where inter-hierarchy comparison is requested. However, we see also the disadvantage of this free binding and we assume that a closer relation between HCLP and the hierarchical constraint solver with inter-hierarchy comparison can improve dramatically the efficiency of the system. The reason of this hypothesis is the similarity of constraint hierarchies arising from different choices of rules in HCLP program. Consequently, such hierarchies, or at least their parts, can be solved in common. Additionaly, the tight integration of HCLP with inter-hierarchy comparison can remove some infinite computations which lock the above approach (see Section 6).

We propose to utilize breadth-first search in HCLP with inter-hierarchy comparison. As we need to explore the whole computational tree, the computational complexity of breadth-first search is similar to the computational complexity of depth-first search. We understand that in general the memory consumptions of both methods are different, but again, all hierarchies appearing during the computation are collected and thus, the final memory consumption of both methods does not differ significantly. The advantage of breadth-first search is that it collects all finite constraint hierarchies even if infinite branches appear in the computational tree. The same does not hold for depth-first search.

The elementary switch from depth-first to breadth-first search does not change the efficiency of the HCLP system, however it enables further improvements. Now, we show how to integrate the process of collecting constraint hierarchies with constraint hierarchical solver. In the following, we expect the existence of hierarchical constraint solver supporting globally-better comparators (for such systems look at [1,3,5,10]) and we extend this solver to support inter-hierarchy comparison.

The idea behind our algorithm uses the following observations. One can assign the error vector $EV_{\theta,H}=[g(\theta,H_1), g(\theta,H_2)\dots]$ to each valuation θ of variables from the constraint hierarchy H (*g* is a combining function). Visibly, it is possible to order all error vectors lexicographically and to show that this ordering is a total ordering.

Observation 1:

The solution set S of hierarchy H contains only those valuations from the set S_0 which has the minimal error vector. Moreover, the error vectors of valuations from the solution set S are equal.

The proof of Observation 1 follows directly from the definition of the solution set of hierarchy and the definition of the globally-better schema (see Section 2).

As a consequence of Observation 1 we can assign the error vector $EV_H=EV_{\theta,H}$, where θ is any valuation belonging to the solution set S of the hierarchy H, to each constraint hierarchy H. This error vector EV_H describes how nearly the constraints from the hierarchy H can be satisfied.

Observation 2:

Let EV_H be the error vector of the hierarchy H and $EV_{H\cup\{c\}}$ be the error vector of the extended hierarchy $H\cup\{c\}$ (c is arbitrary constraint added to the hierarchy H). Then $EV_H \leq EV_{H\cup\{c\}}$ in the lexicographic ordering of error vectors.

Observation 2 says that adding arbitrary constraint to the hierarchy does not "improve" the error vector of the hierarchy. The proof of this observation is rather technical and, because of space constraints, we expect to publish it in a subsequent paper.

Now, we can naturally extend Observation 2 to the inter-hierarchy comparison.

Observation 3:

Let EV_H be the error vector of the hierarchy H and EV_J be the error vector of a hierarchy J. If $EV_H < EV_J$ in lexicographic ordering of error vectors, then $EV_H < EV_{J\cup\{c\}}$ for arbitrary constraint *c*.

Observation 3 enables us to eliminate some partial constraint hierarchies, i.e., branches (subtrees) of the computational tree if any complete constraint hierarchy is found during the breadth-first search (see next section).

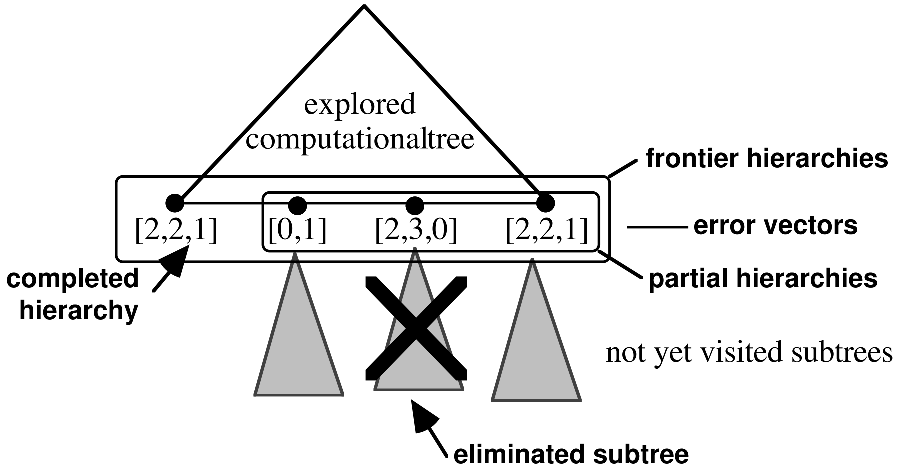
Note also, that the above observations are not in conflict with the nonmonotonic and disorderly aspects of inter-hierarchy comparison presented in [13].

5 The algorithm

As mentioned in above sections, we utilize breadth-first search in our framework for solving HCLP programs using inter-hierarchy comparison. The HCLP goal is resolved in an obvious way. The required (hard) constraints are processed immediately and the soft (preferential) constraints are collected. As soon as the goal is completely reduced in any branch of the computational tree, the computation is interrupted temporarily and the elimination phase starts.

To simplify description of the elimination phase, we classify the collected constraint hierarchies into three categories first. The collected constraint hierarchies which are attached to the leaves of computational branches are called *frontier hierarchies*. We call the constraint hierarchy attached to the leave of the branch terminated with completely reduced goal a *completed hierarchy*. The hierarchies attached to leaves of other branches which are not reduced completely yet are called *partial hierarchies*.

During the elimination phase, the frontier hierarchies are solved using hierarchical constraint solver supporting globally-better comparator. Also the error vectors of all frontier hierarchies are computed. Next, the algorithm eliminates all branches labeled by partial hierarchies which error vector is worse than the error vector of the completed hierarchy. According to Observation 3 we know that the computation in the eliminated branch will not bring better solution (better error vector) than the solution of the completed hierarchy. Thus, the elimination does not influence the final solution. The error vector of the completed hierarchy can be seen as the top estimate of the final solution. Note also that this elimination can cut some infinite branches (see Section 6).



After finishing the elimination, the exploration of the computational tree is restarted. Note, that the solution of all completed hierarchies must be considered in subsequent elimination phases. This means that, if a completed hierarchy H is computed and its solution is better than the solution of previously found completed hierarchies, then the solution of the hierarchy H displaces the previously found solution. Obversely, if the solution of the hierarchy H is worse than the previously found solution, then the solution of the hierarchy H is not included in the current solution (see below the `case` structure of the procedure `distribute_list`).

The algorithm terminates as soon as there is no goal to reduce, i.e., all goals were completely reduced or eliminated respectively.

To simplify the following algorithm scheme we expect that the goal, current valuation of variables and collected soft constraints are encapsulated into one data structure. We also assume that the same data structure with the goal reduced represents the solution.

Algorithm Scheme:

```
solve_goal(Goal,Program)
%% solves Goal using HCLP Program
%% returns list of solutions, i.e., valuations of variables

  GoalsToSolve:=append_to_list(new_list,Goal)
  SolutionOfCompletedHier:=new_list

  while not empty(GoalsToSolve)
    GoalToReduce:=delete_first(GoalsToSolve)
    GoalList:=reduce(GoalToReduce,Program)
    distribute_list(GoalList,GoalsToSolve,SolutionOfCompletedHier)
  end while

  return SolutionOfCompletedHier

end solve_goal
```

```
distribute_list(ListOfGoals,GoalsToSolve,CurrentSolution)
%% distributes list of new goals between list of goals to solve and
   list of solutions found; if any new goal is reduced then do
   elimination

   AnyReduction:=false

   while not empty(ListOfGoals)
     Goal:=delete_first(ListOfGoals)
     if reduced(Goal) then
       AnyReduction:=true
       case of
         :better(Goal,best_of(CurrentSolution))
           CurrentSolution:=append_to_list(new_list,Goal)
                 % displace the previous solution by the new solution

         :not better(best_of(CurrentSolution),Goal)
           % & not better(Goal,best_of(CurrentSolution))
           CurrentSolution:=append_to_list(CurrentSolution,Goal)

         :otherwise
           % the Goal is immediately eliminated
       end case
     else
       GoalsToSolve:=append_to_list(GoalsToSolve,Goal)
     end if
   end while

   if AnyReduction then % elimination
     GoalsToSolve:=delete_worse(best_of(CurrentSolution),GoalsToSolve)
   end if

end distribute_list
```

The call to underlying hierarchical constraint solver is hidden in the test `better` where the error vectors are used to compare solutions. The elimination of not yet reduced goals, which are known to bring worse solution, is done within the procedure `delete_worse`.

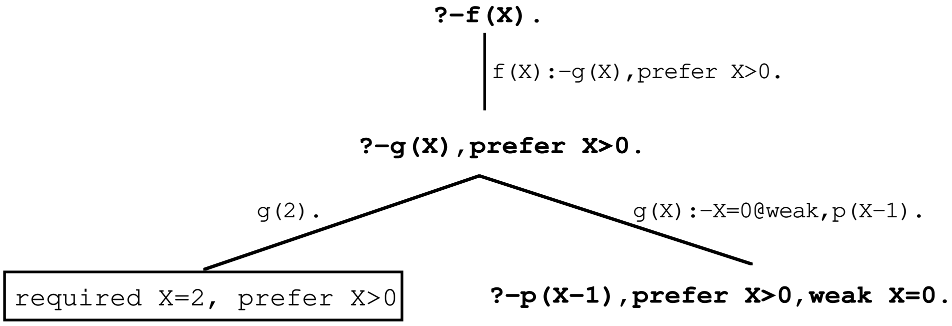
```
delete_worse(BestCompleted,ListOfPartial)

     Goals:=new_list

     while not empty(ListOfPartial)
           G:=delete_first(ListOfPartial)
           if not better(BestCompleted,G)
              Goals:=append_to_list(Goals,G)
           end if
     end while

     return Goals

end delete_worst
```

6 Example

The following example illustrates the computation of the above presented algorithm. We use the domain of integers, the weigted-sum-better comparator (all weights are equal to 1) and the following HCLP program.

```
f(X):-g(X), prefer X>0.
g(2).
g(X):-weak X=0, p(X-1).
p(1).
p(X):-p(X-1).
```

Given the goal `?-f(X)`, the following figure shows the schema of partial computational tree when the first elimination phase occurs, i.e., when a goal in one branch is completely reduced:



The goal in the left branch of the tree is completely reduced and we get the complete hierarchy `required X=2,prefer X>0`. The solution of this hierarchy is `X=2` with the error vector `[0]` (only non-required levels are included in the error vector). The goal `p(X-1)` in the right branch is not reduced yet, and thus we get the partial hierarchy `prefer X>0,weak X=0` with the solution[5] `X=1` and the error vector `[0,1]`. Because the error vector of the partial hierarchy is "worse" than the error vector of the completed hierarchy, the algorithm eliminates the right branch. As there are no other goals to reduce, the algorithm stops and returns the currently found solution `X=2`.

Note that the right subtree, which was eliminated by the algorithm, contains an infinite branch. Therefore, the traditional approach, which collects all constraint hierarchies first, does not terminate at all.

7 Further improvements

The scheme of the algorithm presented in Section 5 offers a lot of opportunities for further improvements. In fact, the presented algorithm corresponds to the well-known branch and bound method. We use breadth-first search to branch the tree of computation. The bound phase called elimination is evoked when the goal in any branch is reduced completely.

It is clear that finding a "good" initial solution can eliminate larger part of the computation tree. However, the breadth-first search is a method of blind search which is not influenced by the "quality" of the (partial) solution. Thus, we propose to use best-first search which is a method of knowledge-driven search. The only change in the algorithm from Section 5 is the displacement of the call `delete_first` in procedure `solve_goal` by the call `delete_best`, which selects the most promising goal to reduce.

The error vector of the hierarchy is the obvious quality indicator for the best-first search. The problem is how to obtain efficiently the error vector of the partial hierarchy. The straightforward approach, which uses the underlying hierarchical constraint solver to solve the constraint hierarchy entirely, does not seem to be reasonable because the evokation of this solver is still time consuming process. So, we suggest to use estimate of the error vector.

Now, it is time to sketch the algorithm for solving constraint hierarchies that we developed in our previous works [1,2,3]. This hierarchical constraint solver has two phases: planning and propagation. The constraint network is constructed during the planning phase, while the valuation of variables is computed during the propagation phase. We will use the features of the planning phase here. The planning phase is well known from hierarchical constraint solvers DeltaBlue [12] and SkyBlue [11]. In [3] we

---

[5] Note, that the hierarchy has no solution if we work with the domain of reals.

generalized the planning phase in such a way that all types of constraints and comparators, including globally-better, are supported. During the planning phase, each constraint is assigned into a constraint cell which labels the node of the constraint network. We distinguish two classes of constraint cells: functional cell and potentially unsatisfied cell. The algorithm guarantees that constraints in functional cells are satisfied during the propagation phase which computes the solution. Moreover, the planning phase of the constraint hierarchical solver can be used naturally to collect soft constraints.

To compute the top estimate of the error vector, we can assume that constraints in pottentially unsatisfied cells are not satisfied. As this planning phase is incremental and thus fast, there is no time penalty to compute the estimate of the error vector.

The changes proposed in this section can improve the efficiency of the algorithm for inter-hierarchy comparison. However, we also mentioned that the expences of computing additional information has to be balanced with the total gain of usage such information.

8 Conclusions and future research

In this paper, we present a basic framework for solving constraint hierarchies using inter-hierarchy comparison within HCLP. The proposed algorithm is based on tight integration with the HCLP interpreter and on elimination of useless branches of computation. The algorithm is built on top of a hierarchical constraint solver supporting globally-better comparators. The elimination of useless branches reduces the space of computation but, on the other hand, this approach demands more attention from the underlying hierarchical constraint solver which is applied to complete constraint hierarchies as well as to partial constraint hierarchies.

The advantage of our approach over the algorithm presented in [6] is the early elimination of unuseful branches of the computation tree which speeds up the computation and reduces the memory consumption. Another benefit is the open architecture of our algorithm that can use arbitrary hierarchical constraint solver supporting globally-better comparators.

There is still a lot of opportunities for further improvements, we show the direction for some of them. Especially adding (semi) incrementality feature to the underlying hierarchical constraint solver can dramaticaly improve the efficiency of the proposed algorithm. Also, the area of empirical investigation of proposed algorithm and its refinements is still open.

We believe that the existence of the open algorithm for inter-hierarchy comparison within HCLP, that is presented in this paper, can further enlarge the area of practical applications of constraint technology and constraint hierarchies, in particular.

Acknowledgments

I would like to thank professor Petr Štěpánek for his continuous support, useful discussions and comments on prerelease version of the paper. This research was supported in part by the Grant Agency of Czech Republic under Grant No. 201/96/0197.

References

[1] Barták, R., A Generalized Algorithm for Solving Constraint Hierarchies, Tech. Report No 97/1, Department of Theoretical Computer Science, Charles University, January 1997

[2] Barták, R., Expert Systems Based on Constraints, Doctoral Dissertation (in Czech), Charles University, Prague, April 1997

[3] Barták, R., A Generalized Framework for Constraint Planning, Tech. Report No 97/9, Department of Theoretical Computer Science, Charles University, June 1997

[4] Borning, A., Duisberg, R., Freeman-Benson, B., Kramer, A., Woolf, M., Constraint Hierarchies, in: *Proceedings of the 1987 ACM Conference on Object Oriented Programming Systems, Languages, and Applications*, pp.48-60, ACM, October 1987

[5] Bouzoubaa, M., Neveu, B., Hasle, G., Houria III: Solver for Hierarchical System, Planning of Lexicographic Weight Sum Better Graph For Functional Constraints, in: *the Fifth INFORMS Computer Science Technical Section Conference on Computer Science and Operations Research*, Dallas, Texas, Jan. 8-10, 1996

[6] Bouzoubaa, M., Hasle, G., Equational Constraint Hierarchies in Constraint Logic Programming Languages: Algorithm for Inter-Hierarchy Comparisons, in: *Proceedings of PACT`96*, pp. 417-426, London, April 24-26, 1996

[7] Borning, A., Maher, M., Martindale, A., Wilson, M., Constraint Hierarchies and Logic Programming, in: *Proceedings of the Sixth International Conference on Logic Programming*, pp. 149-164, Lisbon, June, 1989

[8] Hosobe, H., Miyashita, K., Takahashi, S., Matsuoka, S., Yonezawa, A., Locally Simultaneous Constraint Satisfaction, in: *Principles and Practice of Constraint Programming---PPCP'94 (A. Borning ed.), no. 874 in Lecture Notes in Computer Science*, pp. 51-62, Springer-Verlag, October 1994

[9] Hosobe, H., Matsuoka, S., Yonezawa, A., Generalized Local Propagation: A Framework for Solving Constraint Hierarchies, in: *Principles and Practice of Constraint Programming---CP'96 (E. Freuder ed.), Lecture Notes in Computer Science*, Springer-Verlag, August 1996

[10] Menezes, F., Barahoma, P., Codognet, P., An Incremental Hierarchical Constraint Solver, in: *Proceedings of PPCP'93*, pp. 190-199, Newport, Rode Island, 1993

[11] Sannella, M., The SkyBlue Constraint Solver, Tech. Report 92-07-02, Department of Computer Science and Engineering, University of Washington, February 1993

[12] Sannella, M., Freeman-Benson, B., Maloney, J., Borning,A., Multi-way versus One-way Constraints in User Interfaces: Experience with the DeltaBlue Algorithm, Tech. Report 92-07-05, Department of Computer Science and Engineering, University of Washington, July 1992

[13] Wilson, M., Borning, A., Extending Hierarchical Constraint Logic Programming: Nonmonotonicity and Inter-Hierarchy Comparison, Tech. Report 89-05-04, Department of Computer Science and Engineering, University of Washington, July 1989

[14] Wilson, M., Borning, A., Hierarchical Constraint Logic Programming, TR 93-01-02a, Department of Computer Science and Engineering, University of Washington, May 1993