CHARLES UNIVERSITY
FACULTY OF MATHEMATICS AND PHYSICS

EXPERT SYSTEMS BASED ON CONSTRAINTS

Roman Barták

SUMMARY OF PH.D. DISSERTATION THESIS

Supervisor: Doc. RNDr. Petr Štěpánek, DrSc.

Prague, January 1997

UNIVERZITA KARLOVA
MATEMATICKO-FYZIKÁLNÍ FAKULTA

EXPERTNÍ SYSTÉMY ZALOŽENÉ NA OMEZUJÍCÍCH PODMÍNKÁCH

Roman Barták

Autoreferát doktorandské disertační práce
k získání akademicko-vědeckého titulu doktor.

Obor I-1: Teoretická informatika

Praha, Leden 1997

Disertační práce byla vypracována v rámci doktorandského studia, které uchazeč absolvoval na katedře teoretické informatiky Univerzity Karlovy v letech 1993-1997.

Uchazeč: Mgr. Roman Barták

Školitel: Doc. RNDr. Petr Štěpánek, DrSc.
Katedra teoretické informatiky MFF UK,
Malostranské náměstí 2/25, 118 00 Praha 1

Školící pracoviště: Matematicko-fyzikální fakulta Univerzity Karlovy

Oponenti:

Doc. Ing. Jiří Lažanský, CSc.
Fakulta elektrotechnická, České vysoké učení technické, Praha

Doc. RNDr. Jaroslav Pokorný, CSc.
Matematicko-fyzikální fakulta, Univerzita Karlova, Praha

Autoreferát byl rozeslán dne: ....

Obhajoba se koná dne .... v .... hod. před komisí pro obhajoby disertačních prací oboru I-1 na MFF UK, Ke Karlovu 3, Praha 2, v místnosti č. ....

S doktorandskou disertační prací je možno se seznámit v oddělení pro vědu a výzkum ÚDS MFF UK, Ke Karlovu 3, 120 00 Praha 2.

Doc. RNDr. Petr Hájek, DrSc.
předseda komise pro obhajoby
disertačních prací v oboru teoretická informatika
ÚIVT AV ČR, Pod vodárenskou věží 2, 180 00 Praha 8

1. INTRODUCTION

Constraint hierarchies were introduced for describing over-constrained systems of constraints by specifying constraints with hierarchical strengths or preferences[1]. Constraint hierarchies are widely used in areas like HCLP (Hierarchical Constraint Logic Programming) [9, 25] – an extension of CLP (Constraint Logic Programming) [15] to include constraint hierarchies, CIP (Constraint Imperative Programming) [16] – an integration of declarative constraint programming and imperative object-oriented programming, and graphical user interfaces construction [8]. The major advantage of constraint hierarchies is their declarative expression of preferences or strengths of constraints rather than encoding them in the procedural parts of the language.

In a constraint hierarchy, the stronger a constraint is, the more it influences the solution of the hierarchy. Additionally, constraint hierarchy allows "relaxing" of constraints with the same strength via weighted-sum, least-squares or similar methods. These methods for comparison of valuations according to given constraint hierarchy are called comparators.

Another important aspect of constraint hierarchies is also the existence of efficient satisfaction algorithms. Satisfaction algorithms, in other words constraint hierarchy solvers, can be classified into two groups: algorithms based on refining method and local propagation algorithms. Each group has its advantages and disadvantages but what they have in common is the ad-hoc method used for their construction. Almost all current constraint hierarchy solvers are designed for a specific comparator or for a certain type of constraints.

In my doctoral dissertation I propose to utilize constraint hierarchies in construction of expert systems. In particular, this new approach to expert systems is based on hierarchical constraint logic programming with inter-hierarchy comparison. This decision entails the necessity of effective algorithms for solving constraint hierarchies which support inter-hierarchy comparison. As the current algorithms are not suitable for this intention from various purposes, e.g., they do not support global comparators or they are not effective respectively, we have to design enough effective and general algorithm for solving constraint hierarchies first. Thus, the main contribution of the dissertation is an alternative theory of constraint hierarchies that enables construction of effective algorithms for solving hierarchies. The soundness and completeness of these algorithms are proved there. Also, an instance of effective and general algorithm for solving constraint hierarchies is presented in the dissertation. This algorithm is based on new notions of constraint cell and constraint network. Finally, we present an extension of the proposed algorithm that supports inter-hierarchy comparison.

The algorithm proposed in the dissertation embraces both refining and local propagation concepts, hence it is at once enough efficient and satisfactory general. The algorithm solves constraint hierarchies, even if some constraints must be solved simultaneously, by dividing them into constraint cells as much as possible. By constructing a constraint network it also supports "constraint planning", i.e., the method of smart resatisfying of constraints when a value of one variable is changed. The division of a constraint solving algorithm into two stages, i.e., the planing and the execution stages, is typical for local propagation and we preserve this feature in our algorithm too.

---

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

Actually, we concentrate on the planing stage of the algorithm there as it is independent of the type of constraints and of the chosen comparator.

Although the proposed algorithm shares similar ideas with the DETAIL algorithm [12], it is completely different. While the DETAIL works with equality (functional) constraints and concentrates especially on removing cycles and conflicts from constraint graphs, we mainly focus on support of all types of constraints. Also, the DETAIL allows constraints with different strengths to be in one constraint cell, whereas our algorithm gathers only equally preferred constraints in the constraint cell. It admits the hypothesis that our constraint graphs are more structured, but then, we need more sophisticated execution phase.

This summary is organized as follows. The Part One is dedicated to survey of current research. In Sections 2 and 3, we present a short preview of the original theory of constraint hierarchies followed by a brief catalogue of well-known constraint hierarchy solvers. In Part Two, we explain new results of the dissertation. In Section 4, we analyze limits of current local propagation algorithms for solving constraint hierarchies. In Section 5, we discuss expert systems and we propose expert systems based on constraints. Also, the requirements of these expert systems to underlying system for solving constraint hierarchies are discussed there. We give an alternative formal theory of constraint hierarchies and effective algorithms for solving hierarchies in Section 6. A particular instance of effective algorithm for solving constraint hierarchies is described in Sections 7 and 8. We introduce the notions of a constraint cell and of a constraint network in Section 7 and we also give two sketches of planning algorithms for constructing constraint networks there. In Section 8, we give a preview of the execution algorithm for traversing constraint networks and computing the evaluation of variables. Finally, we discuss the algorithm for solving HCLP goals using inter-hierarchy comparison in Section 9. We conclude with some final remarks on the dissertation and with a survey of future research.

Part One
Survey of current research

2. Constraint Hierarchies

A theory of constraint hierarchies originated in [7]. It allows the user to specify declaratively not only constraints that must hold, but also weaker, so called soft constraints at an arbitrary number of strengths. Weakening the strength of constraints helps to find a solution of previously over-constrained system of constraints. This constraint hierarchy scheme is parameterized by a comparator $\mathcal{C}$ that allows us to compare different possible solutions to a single hierarchy and to select the best ones.

Intuitively, the stronger a constraint is, the more it influences the solution of the hierarchy. Consider, e.g., an over-constrained system of two constraints: `x=0` and `x=1`. The user can attach a preference to both constraints: `x=0@strong` and `x=1@weak`, and the arising constraint hierarchy yields the solution `{x/0}`. This property also enables programmers to specify preferential or default constraints those may be used in case the set of required, so called hard constraints is under-constrained (has more solutions). Moreover, constraint hierarchies allow "relaxing" of constraints with the same preference by applying, e.g., weighted-sum, least-squares or similar methods.

For purposes of the introduction to constraint hierarchies we prefer the earlier definition of constraint hierarchies [9] which is simpler but also a bit different (e.g., it does not support regionally-better comparators) to more recent definition [25].

A *constraint* is a relation over some domain $\mathcal{D}$. The domain $\mathcal{D}$ determines the constraint predicate symbols $\Pi_{\mathcal{D}}$ of the language. A constraint is thus an expression of the form $p(t_1,\ldots t_n)$ where p is an n-ary symbol in $\Pi_{\mathcal{D}}$ and each t_i is a term. A *labeled constraint* is a constraint labeled with a strength (preference), written $c@l$ where c is a constraint and l is a strength. The set of strengths is finite and totally ordered and it is usually given by the user.

A *constraint hierarchy* is a finite set of labeled constraints. Given a constraint hierarchy H, H_0 is a vector of required constraints in H, in some arbitrary order, with their labels removed. Similarly, H_1 is a vector of strongest non-required constraints in H up to the weakest level H_n, where n is a number of non-required levels in the hierarchy H. We also define $H_k=\emptyset$ for $k>n$. Note, that constraints in H_i are stronger (more preferred) than those in H_j for $i<j$.

A *valuation* for a set of constraints is a function that maps free variables in the constraints to elements in the domain $\mathcal{D}$ over which the constraints are defined. A *solution* to a constraint hierarchy is such a set of valuations for the free variables in the hierarchy that any valuation in the solution set satisfies at least the required constraints, i.e., the constraints in H_0, and, in addition, it satisfies the non-required constraints, i.e., the constraints in H_i for $i>0$, at least as well as any other valuation that also satisfies the required constraints. In other words, there is no valuation satisfying the required constraints that is "better" than any valuation in the solution set. Formally:

$$S_0 = \{\ \theta \mid \forall c \in H_0\ \ c\theta \text{ holds}\ \}$$
$$S = \{\ \theta \mid \theta \in S_0\ \&\ \forall \sigma \in S_0\ \neg\ \text{better}(\sigma,\theta,H)\ \},$$

where S_0 is a set of 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*. We insist that *better* is irreflexive and transitive, however, in general, *better* will not provide a total ordering on the set of valuations. We also insist that *better respects the hierarchy*, i.e., if there is some valuation in S_0 that completely satisfies all the constraints through level k, then all valuations in S must satisfy all the constraints through level k:

if $\exists\theta\in S_0\ \exists k>0$ such that $\forall i\in\{1,..,k\}\ \forall c\in H_i\ \ c\theta$ holds

then $\forall\sigma\in S\ \forall i\in\{1,..,k\}\ \forall c\in H_i\ \ c\sigma$ holds.

To define various comparators we first need an *error function* $e(c,\theta)$ that returns a non-negative real number indicating how nearly a constraint c is satisfied for a valuation θ. The error function must have the following property:

$e(c,\theta)=0 \Leftrightarrow c\theta$ holds.

For any domain $\mathcal{D}$, we can use the trivial error function that returns 0 if the constraint is satisfied and 1 if it is not.

Currently, there are two different groups[2] of comparators: locally-better and globally-better comparators. The *locally-better* comparators consider each constraint individually. They are defined by the following way:

locally-better$(\theta,\sigma,H) \equiv_{def}$

$\exists k>0\ \forall i\in\{1,\dots,k-1\}\ \forall c\in H_i\ \ e(c,\theta)=e(c,\sigma)\ \&$

$\exists c'\in H_k\ e(c',\theta)<e(c',\sigma)\ \&\ \forall c\in H_k\ e(c,\theta)\leq e(c,\sigma)$.

We can define a special type of locally-better comparator, *locally-predicate-better* ($\mathcal{LPB}$) comparator to be locally-better using the trivial error function.

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

globally-better$(\theta,\sigma,H,g) \equiv_{def}$

$\exists k>0\ \forall i\in\{1,\dots,k-1\}\ g(\theta,H_i)=g(\sigma,H_i)\ \&$

$g(\theta,H_k)<g(\sigma,H_k)$.

Using globally-better schema, we can define three global comparators, using different combining function g. This comparator triple enables the user to add a positive real number, called *weight*, to each constraint. Weights allow relaxing of constraints with the same strength then. The weight for constraint c is denoted by w_c.

weighted-sum-better$(\theta,\sigma,H) \equiv_{def}$ globally-better(θ,σ,H,g),

where $g(\tau,H_i)=\sum_{c\in H_i} w_c * e(c,\tau)$

worst-case-better$(\theta,\sigma,H) \equiv_{def}$ globally-better(θ,σ,H,g),

where $g(\tau,H_i)=\max_{c\in H_i}\{w_c * e(c,\tau)\}$

least-squares-better$(\theta,\sigma,H) \equiv_{def}$ globally-better(θ,σ,H,g),

where $g(\tau,H_i)=\sum_{c\in H_i} w_c * e^2(c,\tau)$.

---

[2] Recent definition of constraint hierarchies [25] also supports another type of comparator called regionally-better.

Until now, we discussed solution of one constraint hierarchy. Since the comparator compares valuations considering one constraint hierarchy we call this process intra-hierarchy comparison. Sometimes, it is useful to compare valuations with regard to the set of constraint hierarchies. We speak about inter-hierarchy comparison then.

The inter-hierarchy comparison admits more intuitive solutions, e.g., in HCLP [25], which we consider to be an underlying system of proposed expert systems. Fortunately, it is possible to extend the definition of solution considering set of constraint hierarchies in a straightforward manner:

$$S_{0\Delta} = \{ \theta_H \mid H \in \Delta \ \& \ \forall c \in H_0 \ \ c\theta_H \text{ holds } \}$$
$$S_{\Delta} = \{ \theta_H \mid \theta_H \in S_{0\Delta} \ \& \ \forall \sigma_J \in S_{0\Delta} \ \neg \ \text{better}(\sigma_J,\theta_H,\Delta) \ \}.$$

As we compare two valuations with regards to the set of hierarchies it is not possible to use locally-better comparators which consider each constraint individually. Nevertheless, the globally-better comparators can be applied almost directly to inter-hierarchy comparison. The definition of globally-better comparators adapted to inter-hierarchy comparison follows:

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

Note that each valuation is indexed by the hierarchy whose required constraints satisfies. Then, when a combining function is applied to given valuation it uses the level of correspondent hierarchy as a second parameter.

3. CONSTRAINT HIERARCHY SOLVERS

An important aspect of constraint hierarchies is that there are efficient satisfaction algorithms proposed. We can categorize them into the following two approaches:

The *refining algorithms* first satisfy the strongest level, and then weaker levels successively.

The *local propagation algorithms* gradually solve constraint hierarchies by repeatedly selecting uniquely satisfiable constraints.

To illustrate both approaches consider, e.g., the following constraint hierarchy:

`x=y@required, x=z+1@strong, z=1@medium, x=1@weak.`

The refining algorithm first solves the required constraint `x=y` with the result `{x/V,y/V}`, followed by the strong constraint `x=z+1` leading to `{x/Z+1,y/Z+1,z/Z}`. Then, it evaluates the medium constraint `z=1` and gets solution `{x/2,y/2,z/1}`. Finally, it attempts to solve the weak constraint `x=1` but as it conflicts with the assignment generated by the stronger constraints, it remains unsatisfied.

By contrast, the local propagation algorithm first solves the medium constraint `z=1`, then propagates the value `{z/1}` through the strong constraint `x=z+1`, i.e., computes `{x/2,z/1}`, and, finally, through the required constraint `x=y`, i.e., `{y/2,x/2,z/1}`. Note, that the weak constraint `x=1` remains unsatisfied as it was rejected by the stronger constraints.

The refining method is a straightforward algorithm for solving constraint hierarchies as it follows the definition of solution, in particular the property of respecting the hierarchy. It means that the refining method can be used for solving all constraint hierarchies using arbitrary comparator. Its disadvantage is recomputing the solution from scratch everytime a constraint is added or retracted. The refining method was first used in a simple interpreter for HCLP programs [9] and it is also employed in the DeltaStar algorithm [25] and in a hierarchical constraint logic programming language CHAL. We show later (Section 7) that our generalized framework for solving constraint hierarchies covers the refining method.

Local propagation takes advantage of the potential locality of typical constraint networks, e.g., in graphical user interfaces. Basically, it is efficient because it uniquely solves a single constraint in each step (execution phase). In addition, when a variable is repeatedly updated, e.g., by user operation, it can easily evaluate only the necessary constraints to get a new solution. This straightforward execution phase is paid off by a foregoing planning phase that choose the order of constraints to satisfy.

Local propagation is also restricted in some ways. Most local propagation algorithms (DeltaBlue [20], SkyBlue [19], QuickPlan [22], DETAIL [12], Houria [10]) can solve only equality constraints, e.g., linear equations over reals. The exception is the Indigo [6] algorithm for solving inequalities that combines local propagation and refining method. We borrowed the main idea behind the Indigo algorithm, i.e., the propagation of the set of values, to our algorithm. The local propagation algorithms also usually use locally-predicate comparator or its variant respectively. Only Houria III and DETAIL can use globally comparators and Indigo uses metric comparator. Finally, local propagation cannot find multiple solutions for a given constraint hierarchy due to the uniqueness.

Part Two
Results of the dissertation

4. LOCAL PROPAGATION LIMITS IN DEPTH

When we investigated the limits of the local propagation algorithms we identified the following problems:

- solving conflicts among constraints is sometimes inappropriate
- local propagation cannot handle cycles of constraints
- local propagation works only with equality (functional) constraints
- local propagation supports only locally predicate better comparators
- local propagation cannot find multiple solutions.

As the execution phase of the local propagation requires every variable to be computed by just one constraint, the planning phase has to choose among conflicting constraints which bound the variable. However, solving this conflict is sometimes impossible, e.g., when constraints have the same strength (x=1@strong, x=2@strong), and sometimes it is too restrictive, i.e., a weaker constraint (y=1@weak) is disabled (assumed unsatisfied) to enable satisfying of a stronger constraint (y=1@strong) even if the weaker constraint is also satisfied.

The execution phase of the local propagation is a linear process. It means that when a constraint computes the value of one of its variables, the values of all other variables in the constraint have to be known, i.e., the values of these variables have had to be computed by other constraints before. This feature disables solving the set of constraints containing the same variables, e.g., the system of equations (x+y=3, x-y=1). Such a system of constraints corresponds to the cycle in the constraint graph, hence we speak about cycles of constraints. Some local propagation algorithms solve constraint cycles by evoking an external solver [8].

We mentioned the way a constraint is used to compute the value of one of its variables in the above paragraphs. The constraint is assumed there to be a function that computes the value of the output variable from the values of input variables. However, this approach disables many types of constraints like inequalities.

Every constraint, which is used in the execution phase, is completely satisfied while some other constraints are entirely disabled during the planning phase. It implies the application of the predicate type of comparator in the classical local propagation. As every constraint is considered individually in the constraint graph it indicates the usage of the locally-better comparator. Local propagation also cannot find multiple solutions due to the uniqueness of satisfying constraints.

5. EXPERT SYSTEMS AND CONSTRAINT HIERARCHIES

Expert systems belong among the most visible and popular results of research in Artificial Intelligence. Expert (knowledge) system is a software application that simulates behaviour of a human expert. The structure of classical expert system can be expressed by the following equation:

expert (knowledge) system = knowledge base + inference machine, where the knowledge base contains encoded knowledge of a particular human expert(s) and the inference machine determines the manipulation with knowledge. Hereinafter, we will concentrate on the most common expert systems whose knowledge base is made of rules in the following form:

if *premise* then *consequence*.

Thus, we speak about rule-based expert systems.

The inference machine evolves consequences of data from the knowledge base and facts given by user. There are two main approaches to inference machines. The first approach uses forward chaining which subsequently deduces results from given data until the answer to given query is found. The second approach utilizes backward chaining which decomposes the query into subqueries until it can directly validate all subqueries. As the inference machine is an independent part of the expert system there are sometimes developed so called empty expert systems or shells which contain only the inference machine. It is possible to complete the expert system by adding a particular knowledge base to the shell then.

In my doctoral dissertation I propose to use hierarchical constraint logic programming (HCLP) with inter-hierarchy comparison to construct expert systems. The knowledge base is made of the HCLP rules and the constraint hierarchy naturally expresses the uncertain information there. The HCLP interpreter is equivalent to the inference machine then. In my opinion, the expert systems based on constraints are easier to develop and maintain.

In the rest of my dissertation I concentrate on development of effective HCLP interpreter that supports inter-hierarchy comparison. This HCLP interpreter constitutes the kernel of proposed expert systems based on constraints. In particular, I focus on effective and general algorithms for solving constraint hierarchies. Also, the underlying theory that validates the soundness of these algorithms is presented.

6. An Alternative Theory of Constraint Hierarchies

To formally grasp the algorithms for solving constraint hierarchies we built an alternative theory of constraint hierarchies that behaves in a similar way as the original theory described in Section 2. But, the alternative theory provides tools to design effective constraint hierarchy solvers.

First, we redefined the notion of comparator that is now called a hierarchy comparator. The hierarchy comparator is made of level comparators which compare two valuations at one level[3]. The definition of level comparator follows.

Definition 1: (level comparator)

We call the relation $\overset{C}{\leq}$ a level comparator if the following conditions hold (C and C' are sets of constraints, σ, θ and π are valuations and e is an error function):

a) $\sigma \overset{C}{\leq} \theta \wedge \theta \overset{C}{\leq} \pi \Rightarrow \sigma \overset{C}{\leq} \pi$

b) $\forall c \in C\ e(c,\sigma) \leq e(c,\theta) \Rightarrow \sigma \overset{C}{\leq} \theta$

c) $\theta \overset{C \cup C'}{\leq} \sigma \wedge \sigma \overset{C'}{\leq} \theta \Rightarrow \theta \overset{C}{\leq} \sigma$

d) $\sigma \overset{C}{\leq} \theta \wedge \sigma \overset{C'}{\leq} \theta \Rightarrow \sigma \overset{C \cup C'}{\leq} \theta$.

[3] Level is a set of constraints with the same strength in hierarchy.

The conditions a) and b) of the Definition 1 describe expected features of the level comparator, i.e., transitivity and "well behaviour" respectively. The conditions c) and d) of the same definition will help us later to compare valuations at given level by decomposition of the level into "independent" subsets.

Now, we can define relations $\overset{C}{<}$ and $\overset{C}{\sim}$ using level comparator $\overset{C}{\leq}$ in an obvious way:

$$\sigma \overset{C}{<} \theta \equiv_{def} \sigma \overset{C}{\leq} \theta \wedge \neg \theta \overset{C}{\leq} \sigma \qquad \sigma \overset{C}{\sim} \theta \equiv_{def} \sigma \overset{C}{\leq} \theta \wedge \theta \overset{C}{\leq} \sigma.$$

The hierarchy comparator, whose definition follows, compares two valuations according to the constraint hierarchy. The operational semantics of the hierarchy comparator is following. It decomposes the hierarchy into levels and compares valuations at individual levels successively from stronger to weaker levels. Thus, the hierarchy comparator expresses explicitly the idea of respecting the hierarchy.

DEFINITION 2: (hierarchy comparator)

We call the relation $\overset{H}{<}$ a hierarchy comparator if it is defined by the following way (H is a constraint hierarchy, H_l are its levels and σ, θ are valuations):

$$\sigma \overset{H}{<} \theta \equiv_{def} \exists k > 0 \quad \forall l \in \{1, \ldots, k-1\} \quad \sigma \overset{H_l}{\sim} \theta \quad \wedge \sigma \overset{H_k}{<} \theta.$$

Note that the hierarchy comparator is defined in a similar way like the comparator in [25]. Thus, the locally-better and almost all globally-better[4] comparators can be directly redefined using the concept of level and hierarchy comparators.

The following notion of hierarchy satisfier introduces operation for satisfying constraint hierarchies. The hierarchy satisfier is a foundation of effective algorithms for solving constraint hierarchies which are proposed later in this work.

DEFINITION 3: (hierarchy satisfier)

We call the function S a hierarchy satisfier if it is defined by the following way:

$$S(\Theta, H) = \{\sigma \in \Theta \mid \neg \exists \theta \in \Theta \quad \theta \overset{H}{<} \sigma\}.$$

Hierarchy satisfier selects best valuations from a given set of valuations according to constraint hierarchy.

The following lemma shows some interesting properties of hierarchy satisfier.

LEMMA 1: (properties of hierarchy satisfier)

a) $\forall \sigma \in S(\Theta, H) \ (\forall c \in H' \ \ e(c, \sigma) = 0 \Rightarrow \sigma \in S(\Theta, H \cup H') \ )$

b) $\forall \sigma, \theta \in \Theta \ (\ (\sigma \in S(\Theta, H) \ \& \ \sigma \overset{H}{\sim} \theta \ ) \Rightarrow \ \theta \in S(\Theta, H) \ )$

Now, it is straightforward to define the solution of constraint hierarchy using hierarchy satisfier.

4 The worst-case-better comparator cannot be used there as it is not possible to define corresponding level comparator.

DEFINITION 4: (hierarchy solution)

We define the solution S(H) of a hierarchy H by the following way:

S(H)=S(Θ,H),

where *Θ* is a set of all valuations which satisfy all required constraints in H, i.e., constraints in H_0.

Note that the Definition 4 of the hierarchy solution in conjunction with the Definition 3 of the hierarchy satisfier correspond to the definition of the solution set from Section 2. The only difference is the definition of comparators, which is now more restrictive.

In the rest of this section we will concentrate on theoretical foundations of effective algorithms for solving constraint hierarchies. The idea behind these algorithms is the decomposition of the constraint hierarchy into “independent” cells which are successively solved/satisfied using the hierarchy satisfier. Cell is a finite subset of constraint hierarchy containing labeled constraints which have to be solved in tandem. Our goal is to find so sequence of cells $B^1,\ldots,B^n$ as $S(\ldots S(S(\Theta,B^1),B^2),\ldots,B^n)\subseteq S(\Theta,B^1\cup\ldots\cup B^n)$ holds.

The following lemma shows that it is not possible to decompose the constraint hierarchy into arbitrary cells.

LEMMA 2: (invalid decomposition)

There exist hierarchies H and H' such that neither $S(H\cup H')\supseteq S(S(H),H')$ nor $S(H\cup H')\subseteq S(S(H),H')$ is valid.

To find a valid decomposition of the hierarchy into cells that enables us to solve the hierarchy by gradually applying the hierarchy satisfier into cells we define the gradual weakening property of the sequence of cells. The satisfaction of the gradual weakening property guarantees that the nonfulfilment of a constraint is not imposed by satisfying any weaker constraint in a previous cell. Thus, the gradual weakening property describes operationally the property of respecting the hierarchy.

DEFINITION 5: (gradual weakening property)

We say that the sequence of cells $B^1,\ldots,B^n$ satisfies the gradual weakening property if the following implication holds for every $i\in\{1,\ldots n-1\}$:

$$\exists\sigma\in S(S\ldots S(S(\Theta,B^1),B^2),\ldots,B^{i+1})\cup S(\Theta,B^1\cup\ldots\cup B^{i+1})\ \ \exists c@l\in B^{i+1}\ \ e(c,\sigma)>0$$

$$\Rightarrow$$

$$\forall j\le i\ \ \forall c@l\in B^j\ \ \forall c'@l'\in B^{i+1}\ \ l<l'$$

LEMMA 3: (characteristic of gradual weakening property)

Let θ be a valuation in Θ and $B^1,\ldots,B^n$ is a sequence of cells that satisfies the gradual weakening property. If $\sigma\in S(\ldots S(\Theta,B^1),\ldots,B^n)$., $\sigma\overset{B^1\cup\ldots\cup B^n}{\sim}\theta$ and $\forall i\in\{1,\ldots n\}\ \ S(\ldots S(\Theta,B^1),\ldots,B^i)\subseteq S(\Theta,B^1\cup\ldots\cup B^i)$ then the following formulas hold:

a) $\forall i\in\{1,\ldots n\}\ \ \sigma\overset{B^1\cup\ldots\cup B^i}{\sim}\theta$

b) $\forall i\in\{1,\ldots n\}\ \ \theta\in S(\ldots,S(\Theta,B^1),\ldots,B^i)$

The gradual weakening property is a sufficient condition that guarantees the soundness of algorithms for solving constraint hierarchies based on decomposition of the hierarchy into cells and gradual applying the hierarchy satisfier. Note that one of the simplest decompositions satisfying the gradual weakening property is the trivial decomposition into levels. Thus, the refining method (Section 3) is covered by our approach.

THEOREM 4: (soundness)

Let $B^1,\ldots,B^n$ be a sequence of cells satisfying the gradual weakening property. Then the following formula holds:

$$S(\ldots S(\Theta,B^1),\ldots,B^n)\subseteq S(\Theta,B^1\cup\ldots\cup B^n).$$

The Theorem 4 guarantees soundness of the proposed method for solving constraint hierarchies, i.e., all valuations found by gradually applying the hierarchy satisfier into sequence of cells satisfying the gradual weakening property belong into the solution set of the hierarchy.

Sometimes, one needs to find all valuations from the solution set. Hence, we looked for an additional condition(s) which guarantees the completeness of the algorithm. We found that such a condition was linearity of the hierarchy comparator, i.e., the hierarchy comparator totally orders all valuations according to given constraint hierarchy. We call such a comparator a linear comparator.

DEFINITION 6: (linear comparators)

We say that the level comparator $\overset{C}{\leq}$ is a linear level comparator if the following condition holds:

$$\forall\sigma,\theta\;\; \sigma\overset{C}{\leq}\theta\vee\theta\overset{C}{\leq}\sigma.$$

We say that the hierarchy comparator $\overset{H}{\leq}$ is a linear hierarchy comparator if it is defined using linear level comparator.

Note that the following completeness theorem requires other condition to be satisfied, in particular $S(\ldots S(\Theta,B^1),\ldots,B^n)\neq\emptyset$, i.e., the proposed algorithm has to find at least one valuation. The non-emptiness of the solution set is discussed in detail in [25].

THEOREM 5: (completeness)

Let $B^1,\ldots,B^n$ be a sequence of cells satisfying the gradual weakening property, S is a hierarchy satisfier that is defined using linear hierarchy comparator and $S(\ldots S(\Theta,B^1),\ldots,B^n)\neq\emptyset$. Then the following formula holds:

$$S(\ldots S(\Theta,B^1),\ldots,B^n)=S(\Theta,B^1\cup\ldots\cup B^n).$$

The Theorems 3 and 5 provide theoretical foundation of the method for solving constraint hierarchies by decomposition of the hierarchy into cells and gradual application of the hierarchy satisfier. In the following sections we show an instance of this method that uses the notions of constraint cell and constraint network. The algorithms proposed hereinafter are based on tuned version of above presented theory, which enables more refined decomposition and thus the algorithms are more effective.

7. CONSTRAINT NETWORKS AND PLANNING

By addressing problems of the local propagation in Section 4 we made the first step to improve the generality of local propagation algorithms. The second step is the theory presented in Section 6 that certifies the soundness of algorithms for solving constraint hierarchies proposed hereinafter.

To eliminate most of the mentioned problems we introduce *constraint cells*, which can contain more than one constraint and which have different functions. The constraint cell containing more constraints can easily handle conflicts between constraints with the same strength as well as it can naturally manage the constraint cycles (Figure 1). By encapsulating the constraints into a constraint cell we also enable using of more types of comparators including globally-better ones.

We suggest several types of constraint cells. There are "normal" cells, called *functional cells*, containing only one functional constraint that can uniquely compute its output variable(s) from input variables. These cells are the only one enabled by the classical local propagation and the constraints in these cells are known to be completely satisfied independently of the values of the input variables. Then, there are "generalized functional" cells, called *generative cells*, containing constraints which can propagate sets of values of input variables to the set of values of output variable(s). As they can generate a set of values they enable finding multiple solutions. Nevertheless, it is also possible that a constraint in the generative cell is not satisfied according to the values of the input variables. Finally, we introduce *test cells*, which, rather that computing a value of any variable, test the satisfiability of constraint(s) according to given values of input variables.

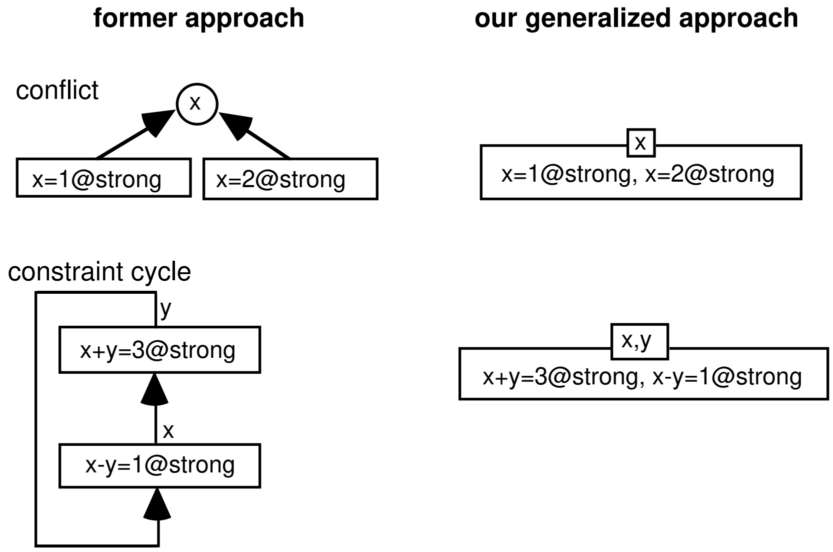


Figure 1 (removing conflicts and constraint cycles)

Individual constraint cells are connected into a constraint network similar to constraints graphs from classical local propagation. This network is created and maintained by the algorithm of the planning phase. Note, that this algorithm is completely independent of a particular type of constraints or used comparator. Due to a more complex structure of the constraint cell we shall also need a more sophisticated algorithm of the execution phase. This algorithm will use a particular comparator and constraint solver to find a solution by tracing the constraint network. Before we proceed to the formal definition of the constraint cell and related notions we depict some examples of constraint cells (we omit the strengths of constraints in the figure). We also outline the propagation of valuations (Section 8) there. Note that every constraint, even the equality, can be in a generative or a test cell.

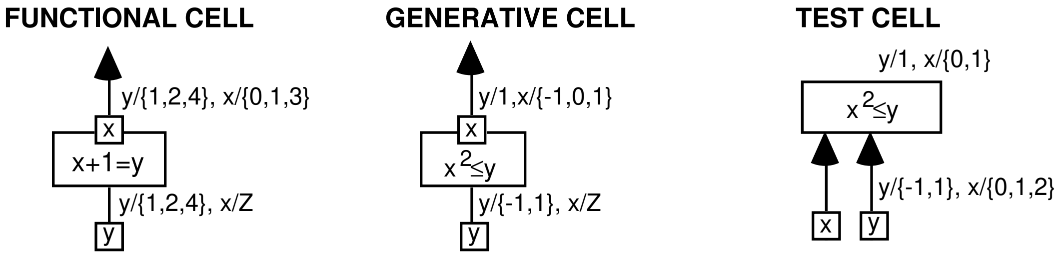


Figure 2 (types of constraint cells)

DEFINITION 7: (constraint cells)

Let C is a finite non-empty set of labeled constraints with the same strength and V is a set of all variables in constraints from C. For arbitrary sets of variables In,Out$\subseteq$V such that In$\cup$Out=V and In$\cap$Out=$\emptyset$ we define a *constraint cell* as a triple (C,In,Out). For every variable v we define a *constraint cell* ({},{},{v}) containing only the output variable v.

We call the sets In and Out from the constraint cell (C,In,Out) *input* and *output variables* respectively.

We also say that constraint cell (C,In,Out) *determinates* each variable from the set Out.

DEFINITION 8: (classification of constraint cells)

We classify constraint cells into the following groups:

- *free variable* ({},{},{v})
- *functional constraint cell* ({c@l},In,Out) such that Out$\neq\emptyset$ and for arbitrary evaluation θ of variables from In there exists a unique valuation σ of variable(s) in Out such that $c\theta\sigma$ holds
- *generative constraint cell* (C,In,Out) such that C$\neq\emptyset$ and Out$\neq\emptyset$ and (C,In,Out) is not functional
- *test* (C,In,$\emptyset$)
- *undecidable constraint cell* is a generative constraint cell or a test

Free variables and functional cells are well known from classical constraint graphs while generative cells and tests are contribution of this work. A constraint in a functional cell is always satisfied but we cannot decide whether constraints in generative cells and tests are satisfied during the planning phase. Thus, we call undecidable both the generative and test cells.

DEFINITION 9: (internal strength)

The *internal strength* of the constraint cell (C,In,Out) is the strength of any constraint in C. The *internal strength* of the constraint cell ({},{},{v}) is ‘free’ which is the strength that is weaker than any other strength of constraints.

DEFINITION 10: (constraint network)

Let H is a constraint hierarchy, i.e., a finite set of labeled constraints, and V is a set of all variables in constraints from H. We call a pair (CC,E) a *constraint network* if the following conditions hold:

1) (CC,E) is a directed acyclic graph with nodes CC and edges E
2) CC is a finite set of constraint cells containing only constraints from H, i.e.,
   $\forall$Cell$\in$CC such that Cell=(C,In,Out)   C$\subseteq$H
3) every constraint from H is located in just one constraint cell, i.e.,
   $\forall$c$\in$H $\exists$! Cell$\in$CC such that Cell=(C,In,Out) & c$\in$C
4) every variable from V is determined by just one constraint cell, i.e.,
   $\forall$v$\in$V $\exists$! Cell$\in$CC such that Cell=(C,In,Out) & v$\in$Out
5) for every constraint cell Cell there exist edges in E directed from constraint cells determining the input variables of Cell, i.e.,
   $\forall$Cell,Cell'$\in$CC
   Cell=(C,In,Out) & Cell'=(C',In',Out') & In$\cap$Out'$\neq\emptyset$ $\Rightarrow$ (Cell',Cell)$\in$E
6) for every undecidable constraint cell there does not exist an upstream constraint cell which has the same or weaker internal strength, i.e., $\forall$Cell$\in$CC
   Cell is undecidable $\Rightarrow$ $\forall$Cell'$\in$CC such that there exists a directed path
   from Cell' to Cell (Cell' is upstream to Cell),
   Cell' has a stronger internal strength than Cell
7) there does not exist "downstream forking" in an undecidable cell directed to other undecidable constraint cells, i.e.,
   $\forall$Cell,Cell'$\in$CC
   Cell and Cell' are undecidable & there does not exist directed path neither from Cell to Cell' nor from Cell' to Cell
   $\Rightarrow$
   $\forall$Cell''$\in$CC such that Cell'' is upstream both to Cell and Cell',
   Cell'' is not undecidable (i.e., it is a functional constraint cell)

The first five points of the constraint network definition are obvious conditions from traditional constraint graphs extended to cover the constraint cells. Thus, the proposed constraint network is a generalization of the former concept of constraint graphs [10,19,20,22]. The new conditions 6 and 7 of the Definition 10 are the contribution of this work. They help us to keep linearity and thus effectiveness of the execution algorithm. As the execution algorithm, computing values of variables, traverses downstream the constraint network, it has to be sure that using a constraint cell to compute its output variables does not disable any stronger constraint later, i.e., downstream the network. The condition 6 preserve this feature. The condition 7 keeps up the linearity of the execution algorithm.

The following figure shows two constraint networks corresponding to the same constraint hierarchy. It implies that there can exist more sound planning algorithms which construct the constraint networks. While the net on the right corresponds to the refining method (all constraints of the same strength are in one cell), the left net is more structured and thus it can more exploit local propagation methods (viz. Section 8).

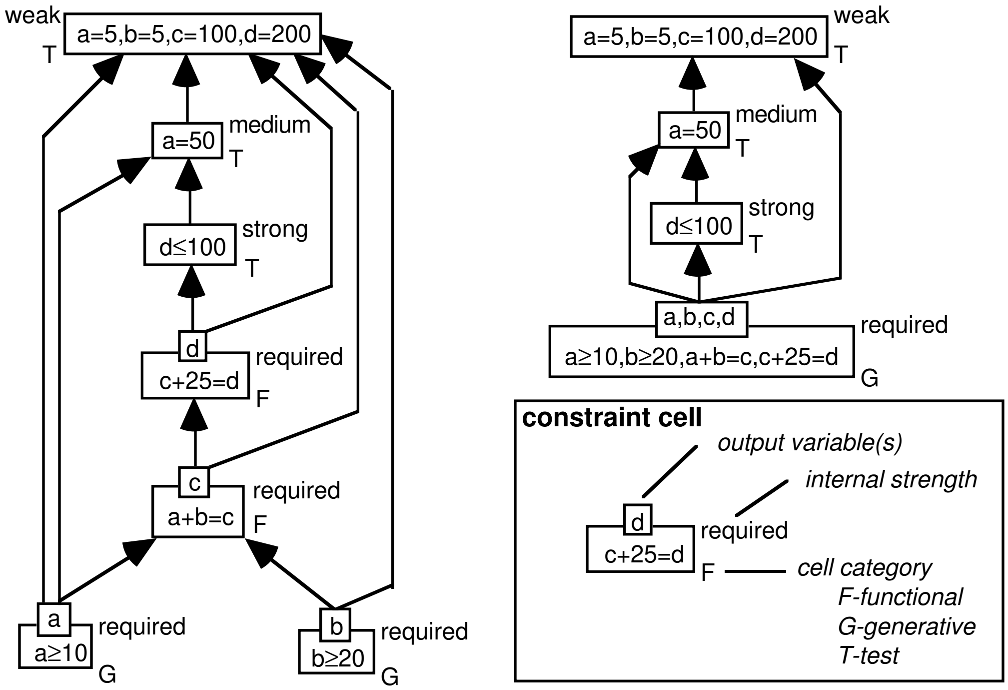


Figure 3 (constraint networks)

The constraint net is incrementally constructed by adding labeled constraints. This stage is usually called planning. We present two planning algorithms in the following paragraphs.

The first algorithm builds constraint nets similar to the right net in the Figure 3. This algorithm behaves in a following way. If there exists a cell with internal strength equal to the strength of the added constraint, then the algorithm adds the constraint into this cell. Otherwise, it creates a new cell containing this constraint. In the second phase the algorithm decides which variables of the added constraint are input and output respectively. Finally, it adds all necessary edges such that all conditions of the Definition 10 are satisfied. Note, that this algorithm does not use the free variable cells. While this planning algorithm is very simple, it requires the execution phase to mimic the refining method and, thus, to be ineffective.

Instead of formal description of the algorithm we give an example of adding a constraint to the net. The following figure shows the process of gradual addition of two constraints into the constraint net (read left to right).

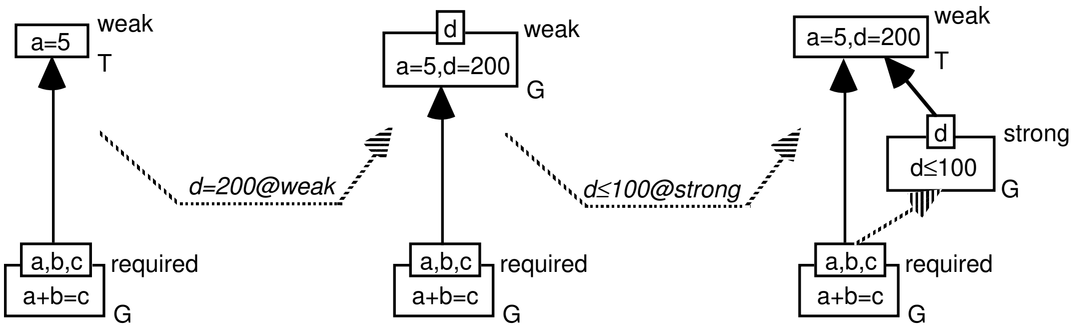


Figure 4 (constraint planning-refining method)

Note, that the shade edge between the required and the strong cell in the rightmost net is not entailed by the definition of the constraint network, nevertheless, this edge is also not explicitly forbidden by the definition. As we expect the execution algorithm to traverse the net from stronger to weaker cells (the refining method), the auxiliary edges can help to better navigate the constraint network.

The sophisticated planning algorithm, that builds structuralized constraint nets like the left net in the Figure 3, keeps the constraint cells as small as possible. This feature of the constraint network is desirable as it enables the execution algorithm to exploit the local propagation as much as possible. Lets us call this planning algorithm a gentle planner contrary to the raw planner that we described above.

The principle of the gentle planner is not complicated. First, the gentle planner tries to add a constraint as a new functional cell. If it does not succeed it adds a constraint as a new generative or test cell. Adding a constraint as a functional cell is almost identical to adding the constraint to a constraint graph using classical local propagation algorithm like DeltaBlue [20]. Nevertheless, we have to keep all conditions from the Definition 10 satisfied. In particular, we have to remove the downstream forking of undecidable cells that could possibly arise after adding a new cell. The following figure sketches the process of removing the downstream forking.

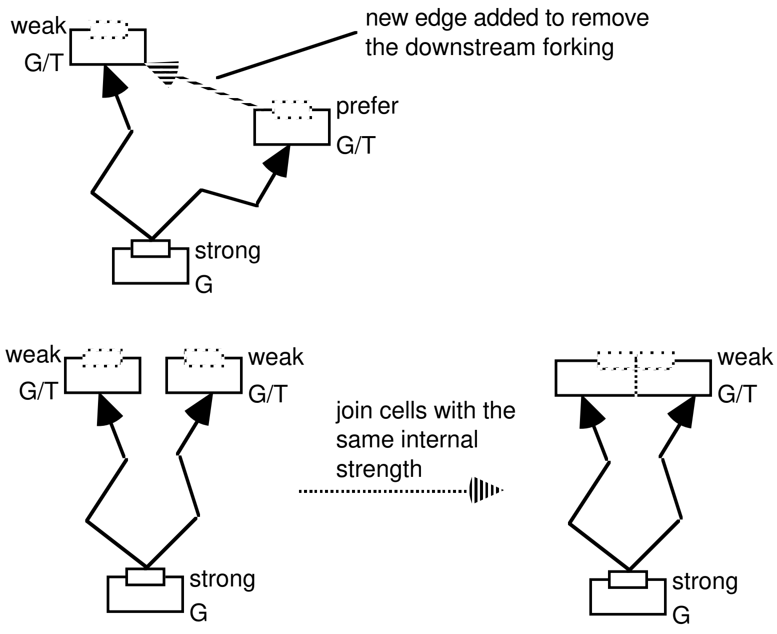


Figure 5 (removing downstream forking)

When it is not feasible to add a constraint as a functional cell, it is added as a generative or test cell. To satisfy the conditions 6 and 7 from the Definition 10 we possibly need to join some cells into one cell. The following figure shows example of adding constraints as generative cells.

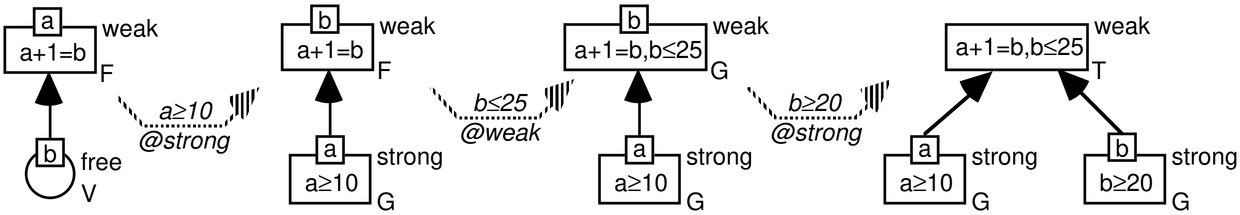


Figure 6 (constraint planning-gentle planner)

Some constraint cells can be deleted from the constraint net during the process of adding a constraint. Constraints from these cells are repeatedly added to the net till all constraints are in the net.

8. EXECUTION PHASE

In this section we will briefly demonstrate one possible algorithm of the execution phase of the generalized constraint hierarchy solver. We expect that many other algorithms of the execution phase can be developed to exploit the proposed structure of the constraint network.

The algorithm presented in this section is based on ideas behind the Indigo algorithm [6]. It means that the algorithm finds a solution of constraint hierarchy containing equality and inequality constraints over reals using locally-error-better comparator (i.e., locally better comparator that uses non-trivial error function). The proposed algorithm propagates set of values or intervals (in the weak version) through the constraint network in a similar way like the Indigo algorithm does. However, our algorithm is able to find multiple solutions contrary to the Indigo.

To demonstrate the algorithm we use the constraint hierarchy whose constraint network is depicted in the Figure 3 (the network on the left). First of all, the algorithm topologically sorts the constraint cells. In this particular case there are two orderings (left to right):

a≥10, b≥10, a+b=c, c+25=d, d≤100, a=50, {a=5,b=5,c=100,d=200}

b≥10, a≥10, a+b=c, c+25=d, d≤100, a=50, {a=5,b=5,c=100,d=200}.

We choose the first sequence and apply directly the Indigo algorithm to the first hexad of constraint cells from this sequence. As every cell in this hexad contains just one constraint we do not need to change the nature of the Indigo algorithm. We get the following partial solution [6]:

a/50, b/{20…25}, c/{70…75}, d/{95…100}.

To get a final solution we choose an arbitrary ordering of constraints in the last constraint cell and apply the Indigo algorithm again. As it is possible to pick the ordering of constraints in the constraint cell it is conceivable to get multiple solutions. The following table shows all final solutions (valuations).

ordering of constraints in the last cell	solution
starts with b=5, e.g., b=5, a=5, c=100, d=200	a/50, b/20, c/70, d/95
starts with c=100 or d=200, e.g., c=100, a=5, b=5, d=200 or d=200, a=5, b=5, c=100	a/50, b/25, c/75, d/100

Note that applying the Indigo algorithm to the constraint cell containing more constraints is justified by using the locally better comparator. When we choose another type of comparator we have to use another method for solving such cells. However, the frame of the algorithm, i.e., the propagation of sets of values through the constraint network, remains the same and thus the modularity [1] of the algorithm is preserved.

In addition to finding multiple solutions, the sketched algorithm of the execution phase can solve the constraint hierarchy more effectively than the Indigo in some cases (the above example is not such a case). While the Indigo sorts constraints according to their strengths only, our algorithm uses topological ordering of constraint cells in the constraint network. Therefore, it is available to place as many as possible functional cells at the beginning of the sequence even if their internal strength is weaker than the strength of other cells and to place as many as possible of the rest functional constraints at the end of the sequence. Then, we can exploit the classical local propagation that is more effective than the Indigo, while the Indigo is used only in the central part of the sequence. The following figure shows such an ordering (the signs F, G and T denote functional, generative and test cells respectively).

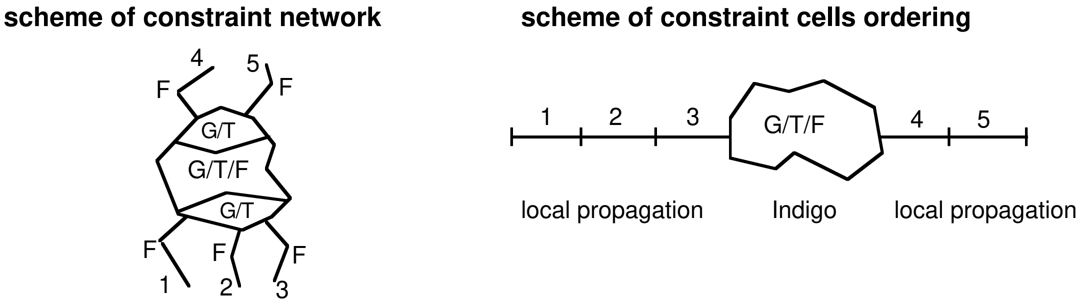


Figure 7 (smart ordering of constraint cells)

Note, that conditions 6 and 7 of the Definition 10 justify the usage of the Indigo algorithm in the central part of the sequence. These conditions ensure that all topological orderings of the constraint cells can be correctly used by the Indigo algorithm that requires stronger constraints to precede the weaker constraints.

9. An Algorithm For Inter-Hierarchy Comparison

The algorithm for solving constraint hierarchies that is described in previous sections does not comprehend inter-hierarchy comparison. Nevertheless, this algorithm tolerates global comparators and thus it is possible to enrich it for inter-hierarchy comparison. We describe such an extension in this section.

A typical HCLP system with inter-hierarchy comparison collects first all constraint hierarchies which arise during the goal reduction using alternative clauses. Then, it solves these hierarchies and compares solutions according to given global comparator [11]. This straightforward method of inter-hierarchy comparison is not effective and sometimes it does not even find the solution. The problem is that the algorithm has to collect all constraint hierarchies even if they do not contribute to solution. Also, if there is an infinite branch in the computation tree, the algorithm cannot collect all hierarchies and so it is not able to find the solution.

In my doctoral dissertation I propose a new algorithm for solving goals in HCLP with inter-hierarchy comparison. This algorithm is more effective and it can solve goals in many cases when the infinite branch occurs. Nevertheless, this algorithm can not engage finding the solution as this is a problem computationally equivalent to the halting problem that is not algorithmically solvable.

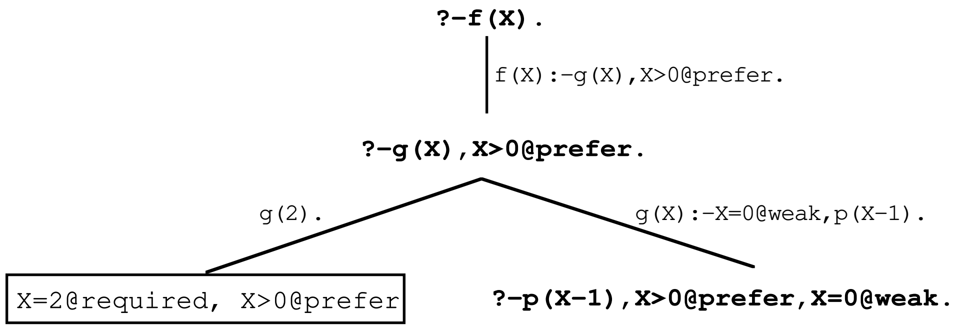
The proposed algorithm is based on breadth-first search that is tuned to computational trees of HCLP goals. This tuning uses the following feature of constraint hierarchies. If the solution of the constraint hierarchy H is better according to a given comparator than the solution of another hierarchy H' then adding constraints to the hierarchy H' (we get hierarchy H^{ext}) does not change this relation, i.e., the solution of the hierarchy H is still better than the solution of the extended hierarchy H^{ext}. This property enables us to eliminate some branches in the computational tree as the following example shows.

Example:

Let us assume the HCLP program:

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

If we solve the goal `?-f(X)` over integers using inter-hierarchy comparison and weighted-sum-better comparator we get the following partial computation tree:



The goal in the left branch is completely reduced and we get the following constraint hierarchy H^L:

`X=2@required, X>0@prefer,`

whose solution is the valuation {X/2}.

The right branch is not completely reduced and we get the partial constraint hierarchy H^R:

`X>0@prefer, X=0@weak.`

whose solution is the valuation {X/1} that satisfies the prefer constraint and minimize the error of the weak constraint.

While the valuation {X/2} satisfies all constraints of "its" hierarchy H^L, the valuation {X/1} does not satisfy the weak constraint of its hierarchy H^R. Thus, we know that by adding other constraints to the hierarchy H^R we will get hierarchies whose solutions are worse than the solution of the hierarchy H^L. Hence, we can leave out reductions in the right branch.
Note also, that by eliminating the right branch we also omit the infinite branch.

The proposed algorithm for solving goals in HCLP using inter-hierarchy comparison exploits breadth-first search with following tuning. When the goal is completely reduced in one branch the algorithm solves the obtained constraint hierarchy H. It also solves all partial constraint hierarchies from remaining branches and it compares the acquired solutions then. If the solution of any partial constraint hierarchy H' is worse than the solution of the "complete" hierarchy H then the algorithm cuts off the branch with the partial hierarchy H'. This approach reduces the computation tree.

It is obvious that early location of a good solution reduces more the computation tree. Thus, we propose to utilize best-first search instead of breadth-first search as a next improvement of the algorithm. The best-first search uses heuristics to find the branch with the "best" solution. We suggest to exploit constraint networks to estimate the worth solution of the hierarchy and to find the most promising branch of the computation tree.

CONCLUSIONS

The doctoral dissertation elaborates the area of effective algorithms for solving constraint hierarchies. The primary argument for studying these algorithms is their usage in newly proposed expert systems based on constraints.

In the dissertation we introduce an alternative theory of constraint hierarchies. This theory provides a general framework for constructing of effective algorithms for solving constraint hierarchies. We addressed some drawbacks of classical local propagation algorithms and, consequentially, we defined more general notions of constraint cell and constraint network. The proposed concept of constraint network is a generalization of former constraint graphs. In particular, we concentrated on the planning phase of the proposed algorithm that constructs the constraint network. We also sketched the algorithm of the execution phase that is based on ideas behind the Indigo algorithm. Finally, we discussed the algorithm for solving HCLP goals using inter-hierarchy comparison.

The presented general framework for solving constraint hierarchies operates as a seed for constructing various constraint hierarchy solvers. In particular, the execution phase is an open area for future research where the experience with solving constraints over various domains is significant. From the theoretical point of view it is possible to explore the sufficient and necessary conditions which ensure the soundness of algorithms for solving constraint hierarchies. Also, we only touched the expert systems based on constraints in this dissertation. So, the other interesting area of future research is formal handling of uncertain information using constraint hierarchies.

ACKNOWLEDGMENTS

I would like to thank my supervisor Petr Štěpánek for his continuous support, useful discussions and valuable comments.

REFERENCES

[1] Barták, R., A Plug-In Architecture of Constraint Hierarchy Solvers, Tech. Report No 96/8, Department of Computer Science, Charles University, December 1996 (to be presented at PACT'97)

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

[3] Barták, R. and Štěpánek, P., Meta-Interpreters and Expert Systems, Tech. Report No 115, Department of Computer Science, Charles University, October 1995

[4] Barták, R. and Štěpánek, P., Mega-Interpreters and Expert Systems, presented at PAP'96, London, April 1996

[5] Benhamou, F. and Colmerauer, A. (eds.), *Constraint Logic Programming-Selected Research*, The MIT Press, Cambridge, Massachusetts, 1993

[6] Borning, A., Anderson, R., Freeman-Benson, B., The Indigo Algorithm, Tech. Report 96-05-01, Department of Computer Science and Engineering, University of Washington, July 1996

[7] 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

[8] Borning, A., Freeman-Benson, B., The OTI Constraint Solver: A Constraint Library for Constructing Interactive Graphical User Interfaces, in: *Proceedings of the First International Conference on Principles and Practice of Constraint Programming*, pp. 624-628, Cassis, France, September 1995

[9] 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

[10] 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

[11] 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

[12] 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

[13] 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

[14] Jaffar, J., Maher, M.J., Constraint Logic Programming: A Survey, in: Journal of Logic Programming 19, pp. 503-581, 1994

[15] Jaffar, J., Lassez, J.-L., Constraint Logic Programming, in: *Proceedings of the 14th ACM Symposium on Principles of Programming Languages*, pp. 111-119, Munich, Germany, January 1987

[16] Lopez, G., Freeman-Benson, B., Borning, A., Implementing Constraint Imperative Programming Languages: The Kaleidoscope'93 Virtual Machine, Tech. Report 94-07-07, University of Washington, July 1994

[17] Meier, M., Brisset, P., Open Architecture for CLP, TR ECRC-95-10, ECRC, 1995

[18] Menezes, F., Barahona, P., An Incremental Hierarchical Constraint Solver, in: [21]

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

[20] 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

[21] Saraswat, V. and Van Hentenryck, P. (eds.), *Principles and Practice of Constraint Programming*, The MIT Press, Cambridge, Massachusetts, 1995

[22] Vander Zanden, Brad, An Incremental Algorithm for Satisfying Hierarchies of Multi-way Dataflow Constraints, Tech. Report, Department of Computer Science, University of Tennessee, March 1995

[23] Van Hentenryck, P., *Constraint Satisfaction in Logic Programming*, Logic Programming Series, The MIT Press, 1989

[24] 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

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