A Generalized Framework for Constraint Planning[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 21 91 42 42
fax: +420-2 53 27 42

Abstract

Constraint hierarchies have been proposed to solve over-constrained systems of constraints by specifying constraints with hierarchical preferences. They are widely used in HCLP, CIP and graphical user interfaces. A declarative expression of preferred constraints and the existence of efficient satisfaction algorithms are the advantages of constraint hierarchies. At present, there exists a lot of relatively independent constraint hierarchy solvers/satisfaction algorithms that could be classified into two categories as refining and local propagation algorithms. While the local propagation algorithms are fast but limited to equality (functional) constraints the more general refining algorithms are not incremental.

In this paper we propose a generalized framework for solving constraint hierarchies, in particular, for constraint planning stage. Our approach is based on ideas of local propagation, however, it suppresses local propagation limits, i.e., restriction to LPB comparator and functional constraints. We introduce a new notion of a constraint cell and we generalize the concept of constraint network here. We also show that the proposed framework covers the refining method as well. Finally, we informally present an algorithm for constraint planning that is an instance of our framework. The proposed algorithm fits in our other concept of plug-in architecture of constraint hierarchy solvers.

Keywords: constraint hierarchy, constraint solvers, constraint planning, constraint network.

---

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 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[3]. 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 [22], i.e., comparison of solutions to two or more constraint hierarchies.

The theory of constraint hierarchies was developed in [6] and it is also described in [8,22,23].

The existence of efficient satisfaction algorithms is another important aspect of constraint hierarchies. Most current satisfaction algorithms, in other words constraint hierarchy solvers, can be classified into two groups: algorithms based on refining method and local propagation algorithms. However, there are also other algorithms, e.g., IHCS [16], which don't fit into any of above mentioned groups.

The refining algorithms solve constraint hierarchy in a straightforward manner by completely satisfying the strongest level first and then weaker levels successively. Thus, the refining method can be used for solving all constraint hierarchies using arbitrary comparator. The refining method was first used in a simple interpreter for HCLP programs [8] and it is also employed in the DeltaStar algorithm [23] and in a hierarchical constraint logic programming language CHAL. We show later (Section 3.3) that our generalized framework for solving constraint hierarchies covers the refining method as well.

The local propagation algorithms take an advantage of the potential locality of typical constraint networks, e.g., in graphical user interfaces. These algorithms gradually solve constraint hierarchies by selecting uniquely satisfiable constraints repeatedly. Many local propagation algorithms were developed for special purposes (DeltaBlue [18], SkyBlue [17], QuickPlan [20], DETAIL [10], Houria [9], Indigo [5]). Our generalized approach to solving constraint hierarchies is also primary based

---

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

3 In [23] , the authors also introduce regionally-better comparators.

on ideas of local propagation which are further generalized to suppress limits of local propagation.

The paper is organized as follows. After brief introduction to constraint hierarchies and overview of algorithms for solving hierarchies in Section 1, we identify limits of current approaches to solving constraint hierarchies in Section 2. In Section 3, we propose a new framework for solving hierarchies, in particular, we concentrate on the planning stage of this framework. We introduce a new notion of a constraint cell and we compare it with the traditional approach to local propagation in Section 3.1. In Section 3.2, we generalize the concept of a constraint network and we justify the soundness of this generalization in Section 3.2.1. The Section 3.3 is dedicated to planning algorithms for construction of constraint networks and we describe there a simple planning algorithm that corresponds to the traditional refining method. The experimental implementation of planning and executing algorithms is described in Section 4. The paper is concluded with some final remarks on the proposed framework and with an Appendix containing formal description of the simple planning algorithm.

2. LIMITS OF CURRENT ALGORITHMS

As we mentioned in the introduction section, most current algorithms for solving constraint hierarchies can be classified into two groups: the refining algorithms and the local propagation algorithms.

The great advantage of the refining method is its generality, so it allows one to solve all types of constraints using arbitrary comparator. We proved this assumption in our plug-in architecture for solving constraint hierarchies [1]. However, the generality of the refining method is paid off by losing effectiveness as each constraint level has to be solved at a clap. In particular, the refining method does not support incremental update of the solution after adding or retracting a constraint respectively.

Contrary, the local propagation is a fast incremental method for resatisfying constraints in hierarchies. Basically, it is efficient because it solves uniquely one selected constraint in every step (executing 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. When we studied the current local propagation algorithms we identified the following limits:

- 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 foregoing 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 (is assumed unsatisfied) to enable satisfying the stronger constraint (y=1@strong) even if the weaker constraint is also satisfied.

The executing 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 are required 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 [7].

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 total function that uniquely computes the value of the output variable from the values of input variables. However, this approach disables other types of constraints like inequalities.

Every constraint, which is used in the executing phase, is completely satisfied while 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 functional constraints.

3. A New Approach To Constraint Planning

By addressing limits of current algorithms we made the first step to improve generality of local propagation as well as to gain efficiency of refining method by encapsulating both approaches into one unified framework. Our framework for solving constraint hierarchies is based on dividing the constraint hierarchy into constraint cells as much as possible. The cells are partially ordered in a constraint network that reflects the relationships among constraints. Finally, the constraint network is traversed by an executing algorithm that propagates valuations through the constraint cells according to the partial order of cells given by the network.

While the propagation of valuations and the partial order of cells defined by the constraint network correspond to the local propagation concept, solving all constraints in a cell at a clap matches the refining method. Hence, the proposed concept of solving constraint hierarchies is enough efficient and satisfactory general at the same time.

Actually, we concentrate on the construction of constraint networks, that we call a planning stage, here.

3.1 CONSTRAINT CELLS

One of the basic parts of our generalized framework for constraint planning is a notion of a constraint cell. Briefly speaking, the constraint cell is a set of equally preferred constraints which are solved in one step during the executing phase. Additionally, we associate a set of output variables with each constraint cell. Then the constraints in a constraint cell determine the output variables of the cell in an obvious manner of local propagation.

DEFINITION 1: (constraint cell)

1) Let C be a finite non-empty set of labeled constraints with the same preference and V be a set of all variables in constraints from C. Let $In, Out \subseteq V$ be arbitrary sets of variables such that $In \cup Out = V$ and $In \cap Out = \emptyset$. We define a *constraint cell* as a triple (C,In,Out). For arbitrary variable v we define a *constraint cell* $(\{\},\{\},\{v\})$ containing only the output variable v.
2) We call the sets In and Out from the constraint cell (C,In,Out) *input* and *output variables* respectively.
3) We also say that the constraint cell (C,In,Out) *determinates* each variable from the set Out.

In an optimum case, each constraint cell contains just one functional constraint. In that case, it is possible to exploit the local propagation as much as possible and the executing phase is as effective as the traditional local propagation (see Section 4). However, by enabling more constraints in the cell one can also easily handle conflicts between constraints with the same strength as well as the constraint cell can naturally manage the constraint cycles. In addition, by encapsulating constraints into a constraint cell we also enable usage of more types of comparators including globally-better ones. Finally, while the traditional local propagation requires all constraints to be functions, i.e., to be able to uniquely compute values of output variable(s) when the values of input variables are given, the proposed constraint cells can contain other types of constraints, e.g., inequalities. The Figure 1 demonstrates the above mentioned advantages of constraint cells over the traditional local propagation.

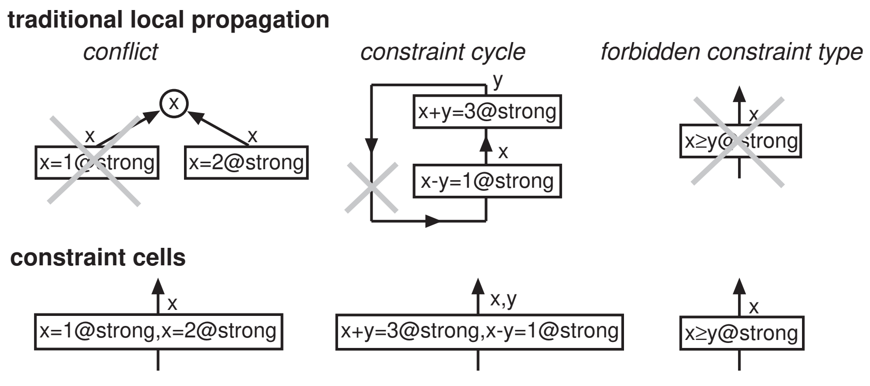

Figure 1 (traditional local propagation vs. constraint cells)

The role of the constraint cell in the constraint network, that we formally introduce in the following section, depends on constraints which are inside the cell and on the distribution of variables into input and output sets of variables. To grasp formally the differences among constraint cells we classify the cells in the following manner.

DEFINITION 2: (classification of constraint cells)

We classify constraint cells into the following groups:

- *free variable cell* ({},{},{v})
- *functional cell* ({c@l},In,Out) such that Out≠∅ and for arbitrary evaluation θ of variables from In there exists a unique valuation σ of variable(s) from Out such that cθσ holds
- *generative cell* (C,In,Out) such that C≠∅ and Out≠∅ and (C,In,Out) is not functional
- *test cell* (C,In,∅)
- *potentially unsatisfied cell* is a generative cell or a test cell

The traditional local propagation enables only free variable cells and functional cells. In fact, instead of the free variable *x*, the traditional local propagation usually uses a special type of constraint in the form `x=user_input` that uniquely determines the value of the variable *x*. Therefore, there are no free variables in traditional constraint graphs, i.e., each variable is uniquely determined by any constraint/functional cell. Thus the propagation of valuations is also unique.

The planning phase of the traditional local propagation decides which constraints are satisfied, and thus they are used for value propagation, and which ones are disabled. The disabled constraints are assumed unsatisfied there. In our generalized approach we use a different mechanism for marking disabled, i.e., potentially unsatisfied, constraints, namely test cells. Thus, the constraints from the functional cells can always be satisfied by binding output variable(s) (see Definition 2) in our approach while the constraints from the test cells can still be satisfied (this is the difference from the traditional local propagation) but the satisfiability is not guaranteed. Namely, there is no degree of freedom to bind output variables in the test cell as there are just no output variables. In other words, all variables from constraints in the test cells are determined by other constraint cells. Note also, that the result of the satisfiability test can be reflected in the value propagation (see Figure 2).

The contribution of this paper is the introduction of generative and test cells. The generative cell is a generalization of the functional cell so that it can contain a non-functional constraint, e.g., inequality, and even a set of constraints. Let us recall that in general the difference between the functional and generative cells depends not only on the constraint itself but also on the distribution of constraint variables into sets of input and output variables respectively. Thus, the cell consisting of the only constraint $\texttt{x}^2\texttt{=y@strong}$ is a functional cell if *x* is an input variable and *y* is an output variable while the similar cell with output variable *x* and input variable *y* is a generative cell (see Definition 2).

Note also, that while each single constraint from a functional cell can always be satisfied, the same does not hold for the generative cell. Assume again the constraint $\texttt{x}^2\texttt{=y@strong}$. Let *x* be the input variable, then for arbitrary value of *x* it is possible to find a unique value of y such that the constraint is satisfied (i.e., the constraint forms a functional cell). However, if y is the input variable (i.e., it is a generative cell), one cannot guarantee the satisfiability of the given constrain, namely, for negative values of y, the constraint is not satisfied, while for non-negative values of y one can find a value of x such that the constraint holds. Note also, that for positive values of y, it is possible to find two values of x which satisfy the constraint. Thus, the value propagation is not unique in the generative cell.

The Figure 2 shows differences among above mentioned types of constraint cells. It also demonstrates the value propagation through the cell.

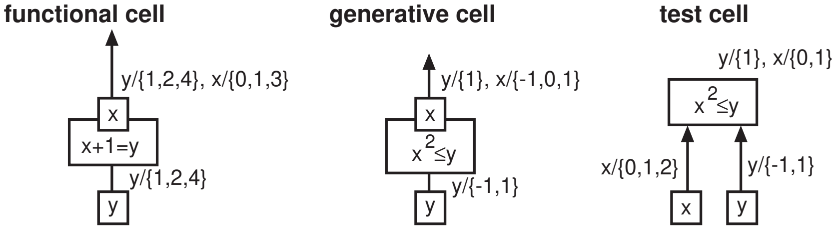

Figure 2 (differences among various constraint cells)

Remind again, that the functional cell propagates values uniquely and the constraint in the functional cell can always be satisfied while the constraints in the potentially unsatisfied cells, i.e., the generative and test cells, are not guaranteed to be satisfied, the value propagation is not unique there and the potentially unsatisfied cells can even influence the values of input variables (see Figure 2). The impact of potentially unsatisfied cells on input variables develops only when the values of input variables are not unique.

In the Figure 2 we omit the strengths of constraints for clarity purpose. Because all constraints in the constraint cell have the same strength (see Definition 1) we can bind this strength with the constraint cell. This so called internal strength of the constraint cell together with the cell type will be the key for partial ordering of cells that is discussed in the following section.

DEFINITION 3: (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.
We denote the internal strength of Cell as i_strength(Cell) and we say that Cell is stronger than Cell' if and only if i_strength(Cell)<i_strength(Cell').

3.2 CONSTRAINT NETWORKS

In this section we concentrate on reasonable decompositions of constraint hierarchies into constraint cells. We also define a partial ordering of cells represented by the constraint network here. This partial ordering of cells guides the sound propagation of valuations through the constraint cells (see Section 3.2.1).

Before formal definition of a hierarchy decomposition we can identify some decompositions in current approaches for solving hierarchies. While the refining method decomposes a given hierarchy into levels of equally preferred constraints, the local propagation atomizes the hierarchy into individual constraints. Both these decompositions and many others are covered by our approach.

DEFINITION 4: (conflict-free hierarchy decomposition)

Let H be a constraint hierarchy, i.e., a finite set of labeled constraints, and V be a set of all variables in constraints from H. We call the finite set CC of constraint cells a *conflict-free decomposition* of the hierarchy H if and only if the following conditions hold:

1) the constraint cells from CC consist only of constraints from H, i.e., $\forall Cell \in CC\ Cell=(C,In,Out)\ \&\ C \subseteq H\ \&\ In \subseteq V\ \&\ Out \subseteq V$
2) every constraint from H is located in just one constraint cell, i.e., $\forall c \in H\ \exists!\ Cell \in CC\ Cell=(C,In,Out)\ \&\ c \in C$
3) every variable from V is determined by just one constraint cell, i.e., $\forall v \in V\ \exists!\ Cell \in CC\ Cell=(C,In,Out)\ \&\ v \in Out$.

The satisfaction of the conditions 1) and 2) of the Definition 4 guarantees that the set CC of cells is really a decomposition of the given hierarchy H. In particular, the satisfaction of the conditions $In \subseteq V$ and $Out \subseteq V$ implies that in the decomposition there do not appear new variables, i.e., variables outside the hierarchy. The satisfaction of the condition 3) of the Definition 4 ensures that each variable is determined by any cell and there is no conflict among cells in the decomposition, i.e., there is no variable that is determined by more than one cell. The following corollary describes this feature formally.

COROLLARY: (no conflicts)

Let CC be a conflict-free decomposition of any constraint hierarchy H. Then the following implication holds:

$$\forall Cell, Cell' \in CC$$
$$(Cell=(C,In,Out)\ \&\ Cell'=(C',In',Out')\ \&\ Cell \neq Cell') \Rightarrow Out \cap Out' = \emptyset$$

Nevertheless, the conflict-free feature of the decomposition does not imply that the decomposition gives an effective solution of the hierarchy by propagation of valuations. Namely, the decomposition, where each variable is determined by a free variable cell and all constraints are in the test cells, is a conflict-free decomposition according to the Definition 4 but this decomposition is of little help for effective constraint hierarchy solving.

In the following paragraphs we concentrate on partial ordering of cells from conflict-free decompositions such that it enables one to effectively solve the constraint hierarchy. Note that some conflict-free decompositions, like the one mentioned in the previous paragraph, cannot be partially ordered according to our requirements, however for every constraint hierarchy there exists at least one conflict-free decomposition that can be represented by the constraint network, i.e., it is possible to partially order the cells in the decomposition.

In what follows, we identify the partial ordering “$<$” of constraint cells with the directed acyclic graph that we call a constraint network. Thus, we write $Cell < Cell'$ if and only if there exist a directed path in the constraint network from Cell to Cell'. Also, we use the denotation G/T for potentially unsatisfied cells in the Definition 5.

DEFINITION 5: (constraint network)

Let CC be a conflict-free decomposition of the hierarchy H. We call the directed acyclic graph (CC,E) with nodes CC and edges E a *constraint network* of hierarchy H if and only if the following conditions hold:

1) 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
2) for every potentially unsatisfied cell there does not exist an upstream constraint cell which has the same or weaker internal strength, i.e.,
$\forall$Cell,Cell'$\in$CC
(Cell is G/T & Cell'<Cell) $\Rightarrow$ i_strength(Cell')<i_strength(Cell)
3) there does not exist "downstream forking" in a potentially unsatisfied cell directed to other potentially unsatisfied cells, i.e.,
$\forall$Cell,Cell',Cell"$\in$CC
Cell,Cell',Cell" are G/T & Cell<Cell' & Cell<Cell"
$\Rightarrow$
(Cell'=Cell" $\vee$ Cell'<Cell" $\vee$ Cell'>Cell")

The Definition 5 describes a constraint network that can be used for value propagation during the executing phase of the constraint hierarchy solving. The meaning of the condition 1) of the Definition 5 is clear as one requires the input variables of the cell to be computed before the value propagation can go through the cell and compute values of output variables. The conditions 2) and 3) of the Definition 5 ensures the correctness of the propagation as one requires the stronger constraints to be prefered to the weaker constraints. The meaning of these conditions is discussed in more detail in the following section. Note, that these two conditions are a new contribution of this paper.

When the constrain network is defined, one may ask whether it is possible to represent an arbitrary constraint hierarchy as a corresponding constraint network. The answer is yes which is not really surprising as we present our approach as a generalization. We justify this claim in Section 3.3 where we describe a simple algorithm for constraint planning. This algorithm decomposes each hierarchy into levels of equally preferred constraints that corresponds to the refining method.

It is also not really surprising that every constraint graph[4] used by traditional local propagation can be represented as a constraint network according to the Definition 5. The satisfied constraint from the constraint graph forms a functional cell in the constraint network while the disabled constraint from the constraint graph is represented as a test cell in the constraint network. Note also, that there exist

4 We use the term "constraint graph" for graphs from the traditional local propagation while the "constraint network" is a generalized notion according to the Definition 5.

constraint hierarchies which cannot be represented by a constraint graph but can be decomposed into a constraint network.

Our approach to constraint networks allows various constraint networks to be constructed for a given constraint hierarchy, from a basic one, that decomposes the hierarchy into levels, to a more structured network, that keeps the cells as small as possible. It depends on the type of used constraints and comparators and on the quality of the planning and executing algorithm which constraint network is chosen. The Figure 3 shows two different constraint networks corresponding to one constraint hierarchy.

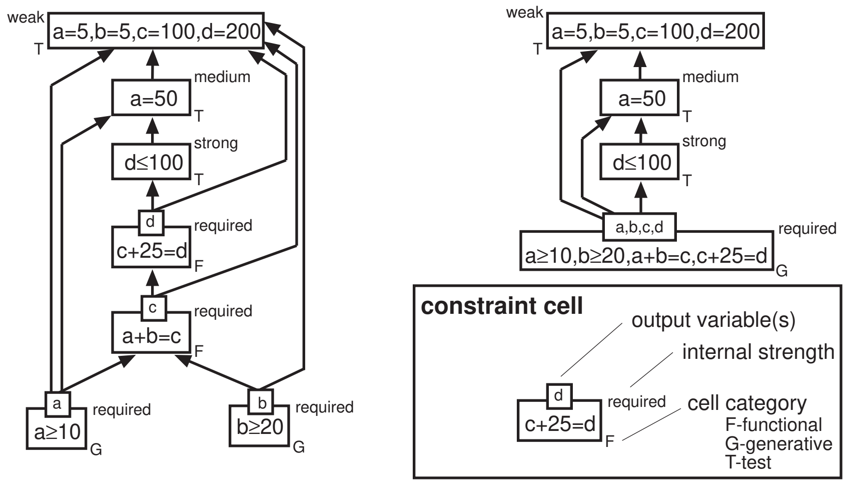

Figure 3 (constraint networks)

3.2.1 A THEORETICAL JUSTIFICATION

A theoretical foundation of our approach originates from ideas described in [11]. In [3] we tuned this theory to the framework proposed in this paper and we proved the soundness and completeness theorems there. A complete formal justification of the proposed framework for solving constraint hierarchies will also be a subject of a separate paper. In the meantime, we satisfy with the following justification.

To keep our framework as general as possible we proceed with the subsequent rule for solving constraint hierarchy:

> *the satisfaction of a stronger constraint is strictly preferred to the satisfaction of an arbitrary number of weaker constraints.*

Thus, what we want to show is that the satisfaction of a constraint does not cause the dissatisfaction of a stronger or equally preferred[5] constraint later on the value

[5] Equally prefered constraints which cannot be satisfied at once should be in one constraint cell so the propagation algorithm can relax them by applying weighted-sum or similar methods. Thus, the globally-better comparators are supported.

propagation. Remind that unsatisfied constraints can be presented only in test or generative cells respectively while the constraints in functional cells can always be satisfied independently of valuation of input variables.

The executing algorithm, that computes the solution to the hierarchy via value propagation, follows the partial order of cells defined by the corresponding constraint network, however it totally orders the cells first to preserve the linearity of the algorithm. The condition 2) of the Definition 5 ensures that there are only stronger cells on upstream path from the potentially unsatisfied cell. So, the only way, how a weaker or equally preferred cell can get before the potentially unsatisfied cell in the total order chosen by the executing algorithm, is that the cell originates from a parallel leaf of the constraint network (the leaf 2a in the Figure 4). The Figure 4 describes all such situations.

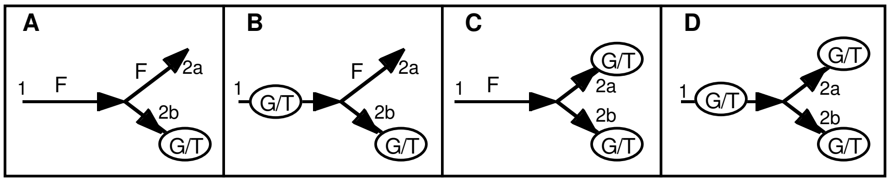

Figure 4 (arrangement of potentially unsatisfied cells in the constraint network)

First, note that the case D of the Figure 4 is prohibited in the constraint networks by the condition 3) of the Definition 5. In the remaining cases A, B and C, it is easy to show that the constraints/cells in the leaf 2a do not influence the satisfaction of the constraints/cells in the leaf 2b, i.e., the satisfaction of any constraint from the leaf 2a does not cause the dissatisfaction of the G/T constraint in the leaf 2b. To confirm this assumption one needs the following features of constraint cells:

1) the functional cells propagate values uniquely
2) the functional cells do not change values of input variables
2) the G/T cells can change the values of input variables only when the values of input variables are not unique.

Note, that when the leaf 1 is empty, the proposition is trivially satisfied. A more detail analysis of the soundness of constraint networks can be found in [3].

3.3 CONSTRAINT PLANNERS

In this section we briefly describe a simple planning algorithm for construction of constraint networks. This algorithm converts arbitrary constraint hierarchy into corresponding constraint network by decomposing the hierarchy into levels of equally preferred constraints. Thus, the existence of the simple planning algorithm confirms our claim that every constraint hierarchy can be represented by a constraint network.

The simple planning algorithm constructs the constraint network by adding labeled constraints incrementally. If there exists a cell with the 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 5 hold. Note, that this algorithm does not use the free variable cells.

While this planning algorithm is very simple, it requires a subsequent execution phase to mimic the refining method. However, such an algorithm of the execution phase is ineffective. A formal description of this simple planning algorithm is given in the Appendix.

According to our experience, there usually exists more than one sound constraint network representing the same constraint hierarchy. It implies that it is possible to develop various planning algorithms which can construct different constraint networks for the same hierarchy. In [3], we present a sophisticated algorithm for constraint planning that keeps the constraint cells as small as possible and thus it enables to more exploit the local propagation.

4. IMPLEMENTATION ISSUES

To test our ideas, we have implemented a prototype system in PROLOG covering the above mentioned algorithms.

We implemented a simple planning algorithm as a part of our plug-in architecture for constraint hierarchy solving [1]. The accomplished experiments with value propagation in arising constraint networks confirmed our belief that using the refining method, especially in case of applying globally-better comparators, was not effective. The ineffectiveness is probably hidden in the necessity to solve all equally preferred constraints at a clap. So, we expect that our approach, that decomposes the hierarchy into smaller cells, can improve the effectiveness of the traditional refining method while preserving its generality.

We also implemented a sophisticated planning algorithm as well as an executing algorithm for value propagation based on Indigo algorithm. Our first results show that the sophisticated planning algorithm behaves as effective as traditional local propagation planning algorithms in cases which the traditional local propagation can handle. Also, our approach enabled a generalization of the Indigo algorithm for value propagation as the executing algorithm can find alternative solutions which the Indigo omitted.

The complete PROLOG source code of the algorithms is available on-line at URL: `http://kti.ms.mff.cuni.cz/~bartak/prolog.html`.

5. CONCLUSIONS

In this paper we present a generalized framework for hierarchical constraint solving, in particular for constraint planning. First, we identified limits of current approaches to solving constraint hierarchies and then we proposed a framework that suppresses these limits. We concentrate on the planning stage of this framework here. We introduced new notions of a constraint cell and a constraint network that naturally expressed the partial ordering of cells. We briefly sketched the soundness of our approach and by describing a simple algorithm for construction of constraint networks we proved that all constraint hierarchies could be represented by a constraint network. We also showed that our framework covered both the refining method and the local propagation as well. Our results are justified not only by a theoretical foundation [3] but also by an implementation work.

The closest work related to our approach is probably [10] that describes the DETAIL algorithm. However, there are some significant differences between DETAIL and our framework. 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 and comparators. Also, the notion of a constraint cell is a bit different in DETAIL where it serves more as a meta-notion denoting a closed part of the constraint graph. DETAIL allows constraints with different strengths to be in one constraint cell, whereas our approach gathers only equally preferred constraints in the constraint cell.

ACKNOWLEDGMENTS

I would like to thank professor Petr Štěpánek for his continuous support, useful discussions and comments on prerelease version of the paper.

REFERENCES

[1] Barták, R., A Plug-In Architecture of Constraint Hierarchy Solvers, to appear in Proceedings of PACT'97, London, April 1997, also available as Tech. Report No 96/8, Department of Computer Science, Charles University, December 1996

[2] Barták, R. and Štěpánek, P., Mega-Interpreters and Expert Systems, presented at PAP'96, London, April 1996, extended version available as Tech. Report No 115, Department of Computer Science, Charles University, October 1995

[3] Barták, R., Expert Systems Based on Constraints, Doctoral Dissertation, Charles University, Prague, April 1997 (english summary available on-line at URL: http://kti.ms.mff.cuni.cz/~bartak/)

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

[5] 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

[6] 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

[7] 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

[8] 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

[9] 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

[10] 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

[11] 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

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

[13] 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

[14] 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

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

[16] Menezes, F., Barahona, P., An Incremental Hierarchical Constraint Solver, in: [19]

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

[18] 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

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

[20] 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

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

[22] 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

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

APPENDIX- A SIMPLE PLANNING ALGORITHM

```
add_constraint(c@l,(CC,E))

%% add a labeled constraint c@l into constraint network (CC,E)
%% return updated constraint network

%% insert constraint to appropriate cell
if ∃Cell∈CC & i_strength(Cell)=l then
   % Cell=(Cs,In,Out)
   Cs:=Cs∪{c@l}          % add c@l to Cell
else
   Cell:=({c@l},{},{})   % create new cell
   CC:=CC∪{Cell}
end if

%% distribute variables and update edges so that the condition 1) of
    the Definition 5 holds
V:=vars(c)
for v∈V do
   if ∃Cell'∈CC & Cell'=(Cs',In',Out') & v∈Out' then
   % variable v is determined by Cell'
      case of
         :(i_strength(Cell')<l)
         % variable v is determined by stronger cell
            % Cell=(Cs,In,Out)
            In:=In∪{v}  % insert v among input variables of Cell
            E:=E ∪ {(Cell',Cell)}  % add corresponding edge
         :(i_strength(Cell')>l)
         % variable v is determined by weaker cell
            % Cell'=(Cs',In',Out')
            % move v in Cell' from output to input variables
            Out':=Out'-{v}
            In':=In'∪{v}
            % remove edges invalidated by moving variable v
            E:=E-{(Cell',C2) | (Cell',C2)∈E & C2=(Cs2,In2,Out2) &
                                    Out'∩In2=∅}
            % Cell=(Cs,In,Out)
            Out:=Out∪{v}  % add v among output variables of Cell
            % add corresponding edges
            E:=E ∪ {(Cell,C2) | C2∈CC & C2=(Cs2,In2,Out2) & v∈In2}
         % if i_strength(Cell')=l, then no change
      end case
   else
   % variable v is not present in the network
      % Cell=(Cs,In,Out)
      Out:=Out∪{v}  % add v among output variables of Cell
   end if
   V:=V-{v}
end for

%% add edges so that the condition 3) of the Definition 5 holds
E:=E∪{(Cell,C2) | C2∈CC & i_strength(C2)=min{i_strength(C)|C∈CC &
                    i_strength(C)>l}}
E:=E∪{(C2,Cell) | C2∈CC & i_strength(C2)=max{i_strength(C)|C∈CC &
                    i_strength(C)<l}}

return (CC,E)

end add_constraint
```