# Solving Finite Domain Constraint Hierarchies by Local Consistency and Tree Search *

S. Bistarelli**[1,2], P. Codognet[3], H.K.C. Hui[4], and J.H.M. Lee[4]

[1] Istituto di Informatica e Telematica, CNR, Pisa, Italy
Stefano.Bistarelli@iit.cnr.it,
[2] Dipartimento di Scienze,
Universitá degli Studi "G. D'annunzio" di Chieti-Pescara, Italy
bista@sci.unich.it,
[3] Department of Computer Science
University of Paris 6, France
Philippe.Codognet@lip6.fr
[4] Department of Computer Science and Engineering,
The Chinese University of Hong Kong, Hong Kong SAR, China
{kchui,jlee}@cse.cuhk.edu.hk

**Abstract.** We provide a reformulation of the constraint hierarchies (CHs) framework based on the notion of *error indicators*. Adapting the generalized view of local consistency in semiring-based constraint satisfaction problems (SCSPs), we define *constraint hierarchy $k$-consistency* (CH-$k$-C) and give a CH-2-C enforcement algorithm. We demonstrate how the CH-2-C algorithm can be seamlessly integrated into the ordinary branch-and-bound algorithm to make it a finite domain CH solver. Experimentation confirms the efficiency and robustness of our proposed solver prototype. Unlike other finite domain CH solvers, our proposed method works for both local and global comparators. In addition, our solver can support arbitrary error functions.

## 1 Introduction

The Constraint Hierarchy (CH) framework [8] is a general framework for the specification and solutions of over-constrained problems. Originating from research in interactive user-interface applications, the CH framework attracts much effort in the design of efficient solvers in the real number domain [1, 17]. To extend the benefit of the CH framework to also discrete domain applications, such as timetabling and resource allocation, the paper takes a step towards a general and efficient finite domain CH solver, based on consistency techniques and tree search.

Central to the paper is the notion of *constraint hierarchy $k$-consistency* (CH-$k$-C), defined using error indicators which are structures isomorphic to the structure of a given

* We thank the anonymous referees for their constructive comments. The work described in this paper was substantially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region (Project no. CUHK4358/02E).

** Part of this research was carried out while the author was visiting the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong SAR, China

CH used for storing the error information of the CH problem (similar notion was defined by Bistarelli *et al.* [4]). We give also an algorithm for enforcing CH-2-C of a CH problem. While classical consistency algorithms [19] aim to reduce the size of constraint problems, our CH-2-C algorithm works by explicating error information that is originally implicit in CH problems. We also suggest ways of utilizing such extracted information to help prune non-fruitful computation in a branch-and-bound searching algorithm, which forms the basis of our finite domain CH solver. We have constructed a prototype of the solver, and performed experiments on a set of randomly generated CH problems that confirm the efficiency and robustness of our proposal.

This paper is a revised and extended version of another by the same authors [3].

The rest of the paper is organized as follows. Section 2 provides necessary background definitions. In Section 3, we present an equivalent redefinition of the CH framework using the notion of error indicators and hierarchy problem, which are central in the definition of constraint hierarchy $k$-consistency and the associated enforcement algorithm in Section 4. In Section 5, we give a constraint hierarchy 2-consistency enforcement algorithm and discuss its complexity. The finite domain CH solver, which has a branch-and-bound backbone, is introduced in Section 6, followed by experimental results in Section 7. Related works are discussed in Section 8 before summarizing the major results and shedding light on possible future direction of research in Section 9.

## 2 Constraint Hierarchies

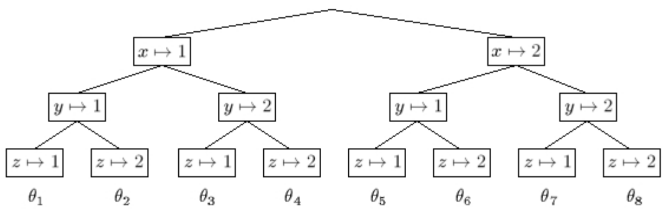
Let $D$ be a constraint domain. A *variable* $x$ is an unknown that has an associated *variable domain* $D(x) \subseteq D$, which defines the set of possible values for $x$. An $n$*-ary constraint* $c$ is a relation over $D^n$. A *labeled constraint* $c^s$ is a constraint $c$ with a *strength* $s \in \{0, \ldots, k\}$. The strengths are totally ordered. Constraints with strength $s = 0$ are *required constraints* (or hard constraints) and those with strength $1 \leq s \leq k$ are *non-required constraints* (or soft constraints). The larger the strength, the weaker the constraint is. In addition, each labeled constraint may be associated with a weight $w$ (for use with the global comparators). A *constraint hierarchy* $H$ is a multiset of labeled constraints. The symbol $H_i$ denotes a set of labeled constraints with strength $s = i$. $H_0$, the *required level*, denotes the set of required constraints which must be satisfied. $H_1, \ldots, H_k$, the *non-required level*, denote the sets of non-required constraints which can be violated but should be satisfied as much as possible. We use an example in Figure 1 to explain CHs in more details. There are three levels in the constraint hierarchy $H$. There are no required constraints in the required level $H_0$. However, there are two

| |
|---|
| $V = \{x, y, z\}$ and $D(x) = D(y) = D(z) = \{1, 2\}$ |
| $H = \{H_0, H_1, H_2, H_3\}$<br>$H_0 = \emptyset$, $H_1 = \{c_1^1 : x > y, c_2^1 : x = 2\}$, and<br>$H_2 = \{c_1^2 : y = 3, c_2^2 : z < y\}$<br>$H_3 = \{c_1^3 : z = 1, c_2^3 : x + y + z > 4\}$ |

Fig. 1: An example of constraint hierarchy.

*strong* constraints $c_1^1$ and $c_2^1$ in $H_1$, two *medium* constraints $c_1^2$ and $c_2^2$ in $H_2$ and two *weak* constraints $c_1^3$ and $c_2^3$ in $H_3$.

A *valuation* $\theta = \{v_1 \mapsto d_1, \ldots, v_n \mapsto d_n\}$ for a set of variables $\{v_1, \ldots, v_n\}$ assigns to each $v_i$ the value $d_i \in D(v_i)$. Let $c$ be a constraint and $\theta$ a valuation. The expression $c\theta$ is the boolean result of applying $\theta$ to $c$. We say that $c\theta$ *holds* if $c\theta$ is *true*. An *error function* $e(c\theta)$ measures how well a constraint $c$ is satisfied by valuation $\theta$. The error function returns non-negative real numbers and must satisfy the property: $e(c\theta) = 0 \Leftrightarrow c\theta$ holds. A *trivial error function* is an error function that gives 0 if $c\theta$ holds and 1 otherwise. The value $e(c\theta)$ returned by an error function is an *error value*. We use $vars(c)$ (or $vars(\theta)$) to denote the set of all variables in constraint $c$ (or valuation $\theta$). The possible valuations for the variables $\{x, y, z\}$ are $\{\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7, \theta_8\}$. Figure 2 gives the error values of all valuations in the complete search tree using the trivial error function. The error values of a valuation $\theta$ are computed for each constraint $(e(c_1^1\theta), e(c_2^1\theta), e(c_1^2\theta), e(c_2^2\theta), e(c_1^3\theta), e(c_2^3\theta))$. Since, for example, $\theta_1$ satisfies $c_1^3$ but violates $c_1^1$, $e(c_1^3\theta_1) = 0$ and $e(c_1^1\theta_1) = 1$ respectively. We can obtain the error values of other valuations similarly. In order to compare values, a number of *comparators* are defined: *locally-better* (*l-b*), *weighted-sum-better* (*w-s-b*), *worst-case-better* (*w-c-b*), and *least-squares-better* (*l-s-b*). We can use these comparators to define *solutions* of CHs [8].




| $\theta$ | Error values for $e(c_j^i\theta)$ | $\theta$ | Error values for $e(c_j^i\theta)$ |
|---|---|---|---|
| $\theta_1$ | $(1, 1, 1, 1, 0, 1)$ | $\theta_5$ | $(0, 0, 1, 1, 0, 1)$ |
| $\theta_2$ | $(1, 1, 1, 1, 1, 1)$ | $\theta_6$ | $(0, 0, 1, 1, 1, 0)$ |
| $\theta_3$ | $(1, 1, 1, 0, 0, 1)$ | $\theta_7$ | $(1, 0, 1, 0, 0, 0)$ |
| $\theta_4$ | $(1, 1, 1, 1, 1, 0)$ | $\theta_8$ | $(1, 0, 1, 1, 1, 0)$ |

Fig. 2: The possible valuations and their error values.

## 3 A Reformulation of Constraint Hierarchies

To facilitate subsequent illustration of the CH local consistency concept, we formulate the CH framework [8] (in particular in the definition of comparators and solution set) using error indicators (as defined in [4]).

We denote an error value by $\xi$, possibly with subscripts. Let $I = \{\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_N\}$ be a poset (partially ordered set), each element $\boldsymbol{\xi}_j$ of which is an *error indicator*. Given a constraint hierarchy $H = \{H_0, \ldots, H_n\}$ where $n$ is the number of non-required levels, and for all $i \in \{0, \ldots, n\}$, $H_i = \{c_1^i, \ldots, c_{k_i}^i\}$ with $k_i$ being the number of constraints in level $i$. An error indicator $\boldsymbol{\xi}_\theta$ of a valuation $\theta$ for a set of variables $V$

is a tuple of error values such that $\boldsymbol{\xi}_\theta = \langle\langle \xi_{\theta}{}_1^0, \ldots, \xi_{\theta}{}_{k_0}^0\rangle, \ldots, \langle \xi_{\theta}{}_1^n, \ldots, \xi_{\theta}{}_{k_n}^n\rangle\rangle$ and $\forall a \in \{0, \ldots, n\}, \forall b \in \{1, \ldots, k_a\}, \xi_{\theta}{}_b^a = e(c_b^a\theta)$ if $vars(c_b^a) \subset V$ and $\xi_{\theta}{}_b^a = 0$ if $vars(c_b^a) \not\subset V$. Error indicators provide a measure of the "badness" of valuations with respect to $H$.

To explain the meaning of the error indicator of a valuation, we use the example in Figure 1 with the trivial error function. If $\theta = \{z \mapsto 2\}$, then $\boldsymbol{\xi}_\theta = \langle\langle\rangle, \langle 0, 0\rangle, \langle 0, 0\rangle, \langle 1, 0\rangle\rangle$. If $\theta = \{x \mapsto 1, y \mapsto 2\}$, then $\boldsymbol{\xi}_\theta = \langle\langle\rangle, \langle 1, 1\rangle, \langle 1, 0\rangle, \langle 0, 0\rangle\rangle$. If $\theta = \{x \mapsto 2, y \mapsto 2, z \mapsto 1\}$, then $\boldsymbol{\xi}_\theta = \langle\langle\rangle, \langle 1, 0\rangle, \langle 1, 0\rangle, \langle 0, 1\rangle\rangle$.

The comparator predicate *better* in the original CH formulation is redefined using a *partial order*, denoted by $\prec$. We define $\prec$ to be irreflexive and transitive over $I$. Hence, it preserves the meaning of *better*. Intuitively, $\boldsymbol{\xi}' \prec \boldsymbol{\xi}''$ means $\boldsymbol{\xi}''$ is "better" than $\boldsymbol{\xi}'$ in $I$. In general, $\prec$ will not provide a total ordering. For convenience, we define $\preceq$ such that $\forall \boldsymbol{\xi}', \boldsymbol{\xi}'' \in I, \boldsymbol{\xi}' \preceq \boldsymbol{\xi}'' \rightarrow (\boldsymbol{\xi}' \prec \boldsymbol{\xi}'') \vee (\boldsymbol{\xi}' = \boldsymbol{\xi}'')$.

We can redefine *l-b* in the original formulation as a partial order $\prec_{l-b}$ as follows. Given any two valuations $\theta$ and $\sigma$, and the corresponding error indicators $\boldsymbol{\xi}_\theta$ and $\boldsymbol{\xi}_\sigma$, $\prec_{l-b}$ is defined as:

$$\begin{aligned}\boldsymbol{\xi}_\theta \prec_{l-b} \boldsymbol{\xi}_\sigma \equiv\ & \exists l > 0 \text{ such that } \forall i \in \{0, \ldots, l-1\},\\ & \forall j \in \{1, \ldots, k_i\}, \xi_{\theta}{}_j^i = \xi_{\sigma}{}_j^i\\ & \wedge \exists a \in \{1, \ldots, k_l\}, \xi_{\sigma}{}_a^l < \xi_{\theta}{}_a^l\\ & \wedge \forall b \in \{1, \ldots, k_l\}, \xi_{\sigma}{}_b^l \leq \xi_{\theta}{}_b^l.\end{aligned}$$

The intuitive meaning of $\boldsymbol{\xi}_\theta \prec_{l-b} \boldsymbol{\xi}_\sigma$ is that valuation $\sigma$ is *locally-better* than valuation $\theta$.

Similarly, we can define *g-b* $\prec_{g-b}$, and its instances *w-s-b* $\prec_{w-s-b}$, *w-c-b* $\prec_{w-c-b}$, and *l-s-b* $\prec_{l-s-b}$ respectively. Given any two valuations $\theta$ and $\sigma$, and the corresponding error indicators $\boldsymbol{\xi}_\theta$ and $\boldsymbol{\xi}_\sigma$:

$$\begin{aligned}\boldsymbol{\xi}_\theta \prec_{g-b} \boldsymbol{\xi}_\sigma \equiv\ & \exists l > 0 \text{ such that } \forall i \in \{0, \ldots, l-1\},\\ & g(\langle \xi_{\theta}{}_1^i, \ldots, \xi_{\theta}{}_{k_i}^i\rangle) = g(\langle \xi_{\sigma}{}_1^i, \ldots, \xi_{\sigma}{}_{k_i}^i\rangle)\\ & \wedge g(\langle \xi_{\sigma}{}_1^l, \ldots, \xi_{\sigma}{}_{k_l}^l\rangle) < g(\langle \xi_{\theta}{}_1^l, \ldots, \xi_{\theta}{}_{k_l}^l\rangle),\end{aligned}$$

where $g$ is a *combining function* for error values:

$$\boldsymbol{\xi}_\theta \prec_{w-s-b} \boldsymbol{\xi}_\sigma \equiv \boldsymbol{\xi}_\theta \prec_{g-b} \boldsymbol{\xi}_\sigma, \text{ where } g(\langle \xi_1^i, \ldots, \xi_{k_i}^i\rangle) \equiv \textstyle\sum_{j \in \{1, \ldots, k_i\}} \xi_j^i,$$
$$\boldsymbol{\xi}_\theta \prec_{w-c-b} \boldsymbol{\xi}_\sigma \equiv \boldsymbol{\xi}_\theta \prec_{g-b} \boldsymbol{\xi}_\sigma, \text{ where } g(\langle \xi_1^i, \ldots, \xi_{k_i}^i\rangle) \equiv \max \xi_j^i \mid j \in \{1, \ldots, k_i\}\},$$
$$\boldsymbol{\xi}_\theta \prec_{l-s-b} \boldsymbol{\xi}_\sigma \equiv \boldsymbol{\xi}_\theta \prec_{g-b} \boldsymbol{\xi}_\sigma, \text{ where } g(\langle \xi_1^i, \ldots, \xi_{k_i}^i\rangle) \equiv \textstyle\sum_{j \in \{1, \ldots, k_i\}} {\xi_j^i}^2.$$

Notice that by definition, all local/global comparators ignore constraints in hierarchy levels greater than or equal to $l$.

We are now ready to define the solution set $S$ of a CH with variables $V$ by:

$$S_0 = \{\theta \mid vars(\theta) = V, \xi_{\theta}{}_i^0 = 0 \text{ for all } i \in \{1, \ldots, k_0\}\} \text{ and}$$
$$S = \{\theta \in S_0 \mid \forall \sigma \in S_0, \boldsymbol{\xi}_\theta \not\prec \boldsymbol{\xi}_\sigma\}.$$

The following lemma gives the monotonicity of the introduced comparators, which are collectively denoted by $\prec_{better}$ and $\preceq_{better}$ in the rest of the paper.

**Lemma 1.** *Given any two error indicators $\boldsymbol{\xi}'$ and $\boldsymbol{\xi}''$. If for all $a, b$ we have ${\xi''}^a_b \leq {\xi'}^a_b$, then $\boldsymbol{\xi}' \preceq_{better} \boldsymbol{\xi}''$.*

Notice that the above lemma lets us compare valuation for both local and global comparators (because the $\preceq_{better}$ order implies all the orders induced from any specific comparator) and for arbitrary error functions.

We also introduce the notion of a *hierarchy problem* which is a CH augmented with error information.

**Definition 1 (Hierarchy Problem and Error Indicator Store).** *A hierarchy problem $P = \langle H, I_H \rangle$ is a constraint problem, where $H$ is a CH with variables $V$ and $I_H$ is a set containing error indicator stores $\boldsymbol{\xi}_{x=d}$ for all variables $x \in V$ and for all $d \in D(x)$. Each $\boldsymbol{\xi}_{x=d}$ is used for keeping an estimate (a lower bound) of the errors of valuations involving $\{x \mapsto d\}$.*

**Definition 2 (Solution of a Hierarchy Problem).** *A valuation $\theta$ is a* solution *of $P = \langle H, I_H \rangle$ if (1) $\theta$ is a solution of $H$ and (2) $\boldsymbol{\xi}_\theta \preceq_{better} \boldsymbol{\xi}_{x=d}$ for all $\boldsymbol{\xi}_{x=d} \in I_H$.*

In other words, solutions of $P = \langle H, I_H \rangle$ are solutions of $H$ which have a "worse" error than the estimates provided in $I_H$. By the definition, the solutions of $H$ always contain those of $\langle H, I_H \rangle$. Equality holds when the error estimates provided in $I_H$ fails to "filter" out any solutions of $H$.

**Theorem 1.** *Consider a CH $H$ and the associated hierarchy problem $P = \langle H, I_H \rangle$, and denote the* solution sets *of $H$ and $P$ by $S_H$ and $S_P$ respectively.*

- *$S_P \subseteq S_H$, and*
- *$S_P = S_H$ if $\boldsymbol{\xi}_\theta \preceq_{better} \boldsymbol{\xi}_{x=d}$ for all $(x \mapsto d) \in \theta$ and $\theta \in S_H$.*

In particular, a hierarchy problem $\langle H, I_H \rangle$ must share the same solution as $H$ if all $\boldsymbol{\xi}_{x=d} \in I_H$ contain only the error value 0 (*i.e.* no error information). This fact is useful in ensuring the correctness of our local consistency algorithm and the completeness of our branch-and-bound solver later.

## 4 Local Consistency in CHs

The classical notion of *local consistency* [19] characterizes when a constraint problem contains non-fruitful values. The main purpose of detecting local inconsistency is thus to remove the inconsistent values from the variable domains and constraints. Hence, the problem is "simpler" to solve when the problem is smaller. However, we adopt a more general notion of local consistency used for SCSP: *"Applying a local consistency algorithm to a constraint problem means explicitating some implicit constraints, thus possibly discovering inconsistency at a local level"* [5]. We adapt this general notion for CH, and define *constraint hierarchy $k$-consistency* (CH-$k$-C).

Before defining CH-$k$-C, we need two operations, $\mathcal{MAX}$ and $\mathcal{MIN}$, on error indicators. Given a CH $H$ with $n$ non-required levels and any two error indicators, $\boldsymbol{\xi}_\theta, \boldsymbol{\xi}_\sigma \in I$, for $H$. $\mathcal{MAX}(\boldsymbol{\xi}_\theta, \boldsymbol{\xi}_\sigma)$ is defined as

$$\langle \langle max({\xi_\theta}^0_1, {\xi_\sigma}^0_1), \ldots, max({\xi_\theta}^0_{k_0}, {\xi_\sigma}^0_{k_0}) \rangle, \ldots, \langle max({\xi_\theta}^n_1, {\xi_\sigma}^n_1), \ldots, max({\xi_\theta}^n_{k_n}, {\xi_\sigma}^n_{k_n}) \rangle \rangle$$

and $\mathcal{MIN}(\boldsymbol{\xi}_\theta, \boldsymbol{\xi}_\sigma)$ is

$$\langle\langle \mathit{min}({\xi_\theta}_1^0, {\xi_\sigma}_1^0), \ldots, \mathit{min}({\xi_\theta}_{k_0}^0, {\xi_\sigma}_{k_0}^0)\rangle, \ldots, \langle \mathit{min}({\xi_\theta}_1^n, {\xi_\sigma}_1^n), \ldots, \mathit{min}({\xi_\theta}_{k_n}^n, {\xi_\sigma}_{k_n}^n)\rangle\rangle$$

where $k_i$ is the number of constraints in level $i$ of $H$.

Given two error indicators, $\mathcal{MIN}$ (or $\mathcal{MAX}$) combines the two indicators by taking the best (or the worst). Obviously $\mathcal{MAX}$ and $\mathcal{MIN}$ are commutative and associative. Thus, it makes sense to write $\mathcal{MAX}\{\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_K\}$ and $\mathcal{MIN}\{\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_K\}$) for any $K > 2$.

Given a CH $H$ with variables $V$. If $x \in V$ and $d \in D(x)$, we define

$$\begin{aligned}&approx_k(x \mapsto d) = \\ &\qquad \mathcal{MAX}\{\mathcal{MIN}\{\boldsymbol{\xi}_\theta \mid vars(\theta) = \{x\} \cup U, (x \mapsto d) \in \theta\} \mid U \subset V, |U| = k-1\}\end{aligned}$$

for any $1 \le k \le |V|$. We call it *$k$-approximation*, which provides an estimate of the "badness" of valuations involving the assignment $x \mapsto d$ for all $m$-ary constraints involving $x$ with $m \le k$. Since the error indicators of all valuations involving $x \mapsto d$ might not be comparable, we can only give an approximation, and $approx_{|V|}(x \mapsto d)$ gives an error estimate involving all constraints in the problem. However, calculating $approx_{|V|}(x \mapsto d)$ is computationally expensive, and $approx_k(x \mapsto d)$ for some small $k < |V|$ gives a more practical approximation.

Referring to the same example in Section 2,

$$\begin{aligned}approx_2(y \mapsto 2) &= \mathcal{MAX}\{\mathcal{MIN}\{\boldsymbol{\xi}_{\{x \mapsto 1, y \mapsto 2\}}, \boldsymbol{\xi}_{\{x \mapsto 2, y \mapsto 2\}}\}, \\ &\qquad\qquad \mathcal{MIN}\{\boldsymbol{\xi}_{\{y \mapsto 2, z \mapsto 1\}}, \boldsymbol{\xi}_{\{y \mapsto 2, z \mapsto 2\}}\}\} \\ &= \mathcal{MAX}\{\mathcal{MIN}\{\langle\langle\rangle, \langle 1,1\rangle, \langle 1,0\rangle, \langle 0,0\rangle\rangle, \langle\langle\rangle, \langle 1,0\rangle, \langle 1,0\rangle, \langle 0,0\rangle\rangle\}, \\ &\qquad\qquad \mathcal{MIN}\{\langle\langle\rangle, \langle 0,0\rangle, \langle 1,0\rangle, \langle 0,0\rangle\rangle, \langle\langle\rangle, \langle 0,0\rangle, \langle 1,1\rangle, \langle 1,0\rangle\rangle\}\} \\ &= \mathcal{MAX}\{\langle\langle\rangle, \langle 1,0\rangle, \langle 1,0\rangle, \langle 0,0\rangle\rangle, \langle\langle\rangle, \langle 0,0\rangle, \langle 1,0\rangle, \langle 0,0\rangle\rangle\} \\ &= \langle\langle\rangle, \langle 1,0\rangle, \langle 1,0\rangle, \langle 0,0\rangle\rangle\end{aligned}$$

The following theorem states that $approx_k(x \mapsto d)$ is monotonically decreasing in $k$.

**Theorem 2.** *If $H$ is a CH with variables $V$, $x \in V$ and $d \in D(x)$, then $approx_{k_2}(x \mapsto d) \preceq_{better} approx_{k_1}(x \mapsto d)$, $\forall 1 \le k_1 \le k_2 \le |V|$.*

By using Lemma 1 we can show that $k$-approximations provide upper bounds for the error indicators of complete valuations for any comparators.

**Theorem 3.** *If $H$ is a CH with variables $V$, $x \in V$ and $d \in D(x)$, then $\boldsymbol{\xi}_\theta \preceq_{better} approx_{|V|}(x \mapsto d) \preceq_{better} approx_k(x \mapsto d)$ for all $1 \le k \le |V|$ and all $\theta$ such that $vars(\theta) = V$ and $(x \mapsto d) \in \theta$, where $\preceq_{better}$ represents any locally/globally better comparator.*

Theorem 3 suggests that $k$-approximations can be used as the basis of the notion of local consistency in CH.

A hierarchy problem $P = \langle H, I_H\rangle$ is *constraint hierarchy $k$-consistent (CH-$k$-C)* if the error indicator stores in $I_H$ explicitly indicate the implicit inconsistency information in all $m$-ary constraints in $H$ where $m \le k$. Formally, we define CH-$k$-C as follows.

**Definition 3 (CH k-Consistency (CH-$k$-C)).** *Given a hierarchy problem $P = \langle H, I_H \rangle$ with variables $V$. $P$ is CH-$k$-C if, for all $\boldsymbol{\xi}_{x=d} \in I_H, \boldsymbol{\xi}_{x=d} \preceq_{better} approx_k(x \mapsto d)$ for some $1 \leq k \leq |V|$.*

The CH-$k$-C condition of $P = \langle H, I_H \rangle$ imposes that the estimated error information of $H$ placed in the error indicator stores in $I_H$ is *at least* as accurate as that provided by $k$-approximations. In addition, explicating the error $P = \langle H, I_H \rangle$ using $k$-approximations makes $P$ CH-$k$-C without changing the solution space of $P$.

**Theorem 4.** *Given a hierarchy problem $P = \langle H, I_H \rangle$ with variables $V$. If each $\boldsymbol{\xi'}_{x=d} \in I'_H$ is defined as follows:*

$$\boldsymbol{\xi'}_{x=d} = \begin{cases} \boldsymbol{\xi}_{x=d} & \textit{if } \boldsymbol{\xi}_{x=d} \preceq_{better} approx_k(x \mapsto d) \\ approx_k(x \mapsto d) & \textit{if } approx_k(x \mapsto d) \preceq_{better} \boldsymbol{\xi}_{x=d} \end{cases}$$

*where $\boldsymbol{\xi}_{x=d} \in I_H$, then the hierarchy problem $P' = \langle H, I_H \rangle$ is (1) CH-$k$-C and (2) shares the same solution set as $P$.*

A simple corollary follows directly from Theorems 1 and 4.

**Corollary 1.** *Given a hierarchy problem $P = \langle H, I_H \rangle$ with variables $V$, and $P' = \langle H, I_H \rangle$ defined so that each $\boldsymbol{\xi'}_{x=d} \in I'_H$ is:*

$$\boldsymbol{\xi'}_{x=d} = \begin{cases} \boldsymbol{\xi}_{x=d} & \textit{if } \boldsymbol{\xi}_{x=d} \preceq_{better} approx_k(x \mapsto d) \\ approx_k(x \mapsto d) & \textit{if } approx_k(x \mapsto d) \preceq_{better} \boldsymbol{\xi}_{x=d} \end{cases}$$

*where $\boldsymbol{\xi}_{x=d} \in I_H$. Denote the solution sets of $H$, $P$, and $P'$ by $S_H$, $S_P$, and $S_{P'}$ respectively.*

$$S_H = S_P \Leftrightarrow S_H = S_{P'}$$

## 5 A CH-2-C Enforcement Algorithm

Arc-consistency algorithm is a common and practical technique to detect local inconsistency in classical CSPs [2, 15]. We design and implement an algorithm to enforce CH-2-C. The purpose of the CH-2-C algorithm is to explicate and place in $I_H$ the implicit error information in a CH that is otherwise not visible. Such an algorithm is given in Figure 3. The subroutines **ch1c_pri** and **ch2c_pri**, in Figures 4 and 5 respectively, are responsible for handling unary and binary constraints respectively. The CH-2-C algorithm ensures that all error indicator stores $\boldsymbol{\xi}_{x=d}$ are updated to reach $approx_2(x \mapsto d)$.

Consider a general CH of $n_c$ labeled constraints with $n_v$ number of variables. In addition, the size of the largest variable domain is of $n_d$. The time complexity of the subroutine **ch1c_pri** is simply of $O(n_d)$, since the only repeating operations, lines 4 to 6 in Figure 4, are placed inside a single loop. These operations are repeated until each element in a variable domain is tested. However, the time complexity of the subroutine **update** (Figure 6) is of $O({n_d}^2)$. Therefore, in the worst case, the time complexity of the subroutine **ch2c_pri** is of $O({n_d}^2)$ as shown in Figure 5. Lines 3 to 5 in the pseudocode of the CH-2-C algorithm are the operations for checking constraints as shown in Figure 3. Since these operations should repeat until all the constraints are considered, the time complexity should be of $O(n_c {n_d}^2)$.

**Algorithm 1:** The CH-2-C algorithm.

**ch2c**($H$, $V$, $D$, $I_H$)
**begin**
  **for** $l \leftarrow 1$ *to* $n$ **do**
    **for** $k \leftarrow 1$ *to* $|H_l|$ **do**
      **let** $c$ be the $k^{th}$ constraint in $H_l$;
      $I_H \leftarrow$ **ch1c_pri**($c$, $l$, $k$, $D$, $I_H$);
      $I_H \leftarrow$ **ch2c_pri**($c$, $l$, $k$, $D$, $I_H$);
  **return** $I_H$;
**end**

Fig. 3: The CH-2-C algorithm.

Since an error indicator is a tuple which stores error values of the corresponding constraints, the space complexity for each error indicator is of $O(n_c)$. The memory requirement of the CH-2-C algorithm depends on the number of error indicator stores in $I_H$. Therefore, we require $n_v n_d$ error indicators. The space complexity of the CH-2-C algorithm is simply of $O(n_v n_d n_c)$ in the worst case.

Notice that some better local consistency algorithms could be defined when considering only a specific comparator (see for instance [4] for specific operators dealing with *l-b*).

## 6 A Branch-and-Bound Finite Domain CH Solver

The simplest way to find the solution set of a CH is to construct the complete search tree for the problem, so that we can calculate and compare the error values of each valuation. However, traversing the complete search tree and comparing all the valuations are tedious and time-consuming. We propose to combine the CH-2-C and the branch-and-bound algorithms so as to prune non-fruitful branches of the search tree.

The input to our solver is a hierarchy problem $P = \langle H, I_H \rangle$, in which $I_H$ contains *no* error information. In other words, the error indicator stores in $I_H$ contain only the error value 0. The backbone of our solver is a standard branch-and-bound algorithm, since CH-solving is an optimization problem. A branch-and-bound algorithm always maintains the set of potential best solutions collected so far. The idea is to invoke the CH-2-C algorithm at each node in the search tree, hoping that the overhead in the CH-2-C algorithm can be more than compensated by the pruning that can take place. The correctness and completeness of this step is ensured by Corollary 1, so that maintaining CH-2-C will not change the solution space of the hierarchy problem and the associated CH. At each CH-2-C tree node, before search proceeds down a selected branch corresponding to a variable assignment, say $x \mapsto d$, the solver tries to verify if $\boldsymbol{\xi}_{x=d}$ in $I_H$ of that tree node is not worse than the error indicator of each potential solution. If that is the case, search proceeds; otherwise, there is no point to explore the selected

**ch1c_pri**($c$, $l$, $k$, $D$, $I_H$)
**begin**
  **if** $|vars(c)| = 1$ **then**
    **let** $\{x\} = vars(c)$;
    **for** *each* $d \in D(x)$ **do**
      **let** $\theta = \{x \mapsto d\}$;
      **let** $\boldsymbol{\xi} = \boldsymbol{\xi}_{x=d} \in I_H$;
      **if** $\xi_k^l < e(c\theta)$ **then** $\xi_k^l \leftarrow e(c\theta)$;
  **return** $I_H$;
**end**

Fig. 4: A subroutine to check unary constraints.

branch any further, and search is backtracked to try another branch. When a leaf node is reached, we compare the error indicator $\boldsymbol{\xi}$ of the valuation associated with the leaf node against the error indicators of all the collected solutions. If the error indicator of any collected solution is worse than $\boldsymbol{\xi}$, then the collected solution will be replaced by the current valuation.

Our CH-2-C algorithm ensures that each error indicator store $\boldsymbol{\xi}_{x=d}$ is $approx_2(x \mapsto d)$. By Theorem 3, the error indicator of every complete valuation involving assignment $x \mapsto d$ must be worse than $approx_2(x \mapsto d)$. If at a search node, $\boldsymbol{\xi}_{x=d}$ is worse than the error indicators of each potential solution collected so far, there is no point to search on since all the possible valuations down that branch must be worse than the potential solutions. The details of our finite domain CH solver is shown in Figure 7, which is a simple adaptation of a basic branch-and-bound solver with the CH-2-C algorithm. The numbered lines give the backbone of the algorithm, while the unnumbered lines are new additions to enable CH-2-C enforcement. The algorithm use as parameters the constraints in $H$ and and the stores in $I_H$, the variables $V$ and the domain $D$. It also needs the set of assignments $S_0$ satisfying constraints in $H_0$, and the corresponding set of error indicators $I_{S_0}$. The algorithm is also parametric w.r.t. the type of comparator we want to use ($\prec_{better}$).

Although CH-2-C encompasses also crisp notions of node and arc consistency, we employ classical algorithms [19] for processing the required constraints in $H_0$ (lines 1) for performance reasons. Lines 5 to 13 deal with the case of a leaf node. Here there is a call to subroutine **cal_error_value** that computes the error $e(c\theta)$ for each $\theta$. The CH-2-C algorithm is invoked between lines 13 and 14. Lines 14 to 17 perform the basic variable instantiation (or searching) recursively. The call to the subroutine **go** determines whether the error indicator store of the variable assignment of the selected branch in $I_H$ of the current node is not worse than the error indicator of each of the collected solutions so far.

**ch2c_pri**($c$, $l$, $k$, $D$, $I_H$)
**begin**
    **if** $|vars(c)| = 2$ **then**
        **let** $\{x, y\} = vars(c)$;
        *# Update each* $\boldsymbol{\xi}_{x=d_x} \in I_H$
        $I_H \leftarrow$ **update**($x$, $y$, $c$, $l$, $k$, $D$, $I_H$);
        *# Update each* $\boldsymbol{\xi}_{y=d_y} \in I_H$
        $I_H \leftarrow$ **update**($y$, $x$, $c$, $l$, $k$, $D$, $I_H$);
    **return** $I_H$;
**end**

Fig. 5: A subroutine to check binary constraints.

## 7 Experimental Results

We compare the performance of our proposed solver with generate-and-test, basic branch-and-bound, and the reified constraint approach by Lua (the Lua's solver hereafter) [16]. DeltaStar is only a theoretical framework [11], and clp(FD,S) cannot in the current implementation deal with hierarchies. Since both Lua's solver and ours are based on a branch-and-bound backbone, we first implement a solver engine $S_g$, which searches using ILOG's default *goal* definition, in ILOG Solver 4.4 in a generate-and-test fashion. In order to provide a basic Branch-and-Bound solver (without CH-2-C enforcement) for comparison, we define an alternative ILOG goal to obtain $S_b$. Our proposed solver $S_c$ is obtained by implementing additional functions and an alternative *goal* definition $G_c$ in $S_g$. While the input to our solvers is a CH, the input to Lua's solver $S_r$ ("$r$" stands for "reified constraint") is a CSP with reified constraints for implementing a specific comparator and error function. Our comparison ensures *fairness* since all four solvers share the same backbone.

Our experiments are conducted on Sun Ultra 5/400 workstations with 256MB RAM. We record the execution time taken by $S_g$, $S_b$, $S_c$, and $S_r$ to find the solution set of each problem instance using a particular comparator, denoting these timings $t_g$, $t_b$, $t_c$, and $t_r$. For each problem instance and comparator, we compute three ratios: $t_g/t_c$, $t_b/t_c$, and $t_r/t_c$. Each number in the following tables corresponds to the average of the same type of ratios for fifteen problem instances in a particular problem set $P_i$ and a particular comparator. The columns on the left compare $S_g$ and $S_c$, while the ones in the middle compare $S_b$ and $S_c$, and the ones on the right compare $S_r$ and $S_c$ (only for global comparators). Our 3-part experiments test the effect of *variable domain size*, *number of variables*, and *number of hierarchy levels* on the performance of our proposed solver. In each part, four sets of CHs: $P_1$, $P_2$, $P_3$, and $P_4$, each of which contains 15 problem instances, are generaged *randomly*. All problem instances have no hard constraints to make them more "difficult" to solve.

**update**($x$, $y$, $c$, $l$, $k$, $D$, $I_H$)
**begin**
  **let** $\xi_{min}$ be an error value;
  **for** *each* $d_x \in D(x)$ **do**
    $\xi_{min} \leftarrow \infty$;
    **for** *each* $d_y \in D(y)$ **do**
      **let** $\theta = \{x \mapsto d_x, y \mapsto d_y\}$;
      **if** $e(c\theta) < \xi_{min}$ **then** $\xi_{min} \leftarrow e(c\theta)$;
    **let** $\boldsymbol{\xi} = \boldsymbol{\xi}_{x=d_x} \in I_H$;
    **if** $\xi_k^l < \xi_{min}$ **then** $\xi_k^l \leftarrow \xi_{min}$;
  **return** $I_H$;
**end**

Fig. 6: A subroutine to update error indicator stores.

In the first part, the number of variables and the number of hierarchy levels are fixed ($|V| = 5$, $H = \{H_0, H_1, H_2\}$, $|H_0| = 0$, and $|H_1| = |H_2| = 5$) across all instances, while problems in the same set share a specific domain size: $P_i$ has domains of size $10i$ for $i \in \{1, 2, 3, 4\}$.

| | $t_g/t_c$ (Mean) | | | | $t_b/t_c$ (Mean) | | | | $t_r/t_c$ (Mean) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| CHs | *w-s-b* | *w-c-b* | *l-s-b* | *l-b* | *w-s-b* | *w-c-b* | *l-s-b* | *l-b* | *w-s-b* | *w-c-b* | *l-s-b* |
| $P_1$ | 8 | 5 | 7 | 10 | 6 | 4 | 6 | 7 | 5 | 4 | 5 |
| $P_2$ | 36 | 15 | 37 | 13 | 18 | 22 | 19 | 9 | 9 | 19 | 9 |
| $P_3$ | 267 | 67 | 261 | 171 | 121 | 47 | 123 | 31 | 113 | 42 | 115 |
| $P_4$ | 385 | 72 | 342 | 76 | 37 | 35 | 39 | 23 | 17 | 27 | 18 |

In the second part, the variable domain size and the number of hierarchy levels are fixed ($|D(x)| = 5$ for all variables $x$, $H = \{H_0, H_1, H_2\}$, $|H_0| = 0$, and $|H_1| = |H_2| = 5$) across all instances, while problems in the same set share a specific number of variables: $P_i$ has $2(i + 1$ variables for $i \in \{1, 2, 3, 4\}$.

| | $t_g/t_c$ (Mean) | | | | $t_b/t_c$ (Mean) | | | | $t_r/t_c$ (Mean) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| CHs | *w-s-b* | *w-c-b* | *l-s-b* | *l-b* | *w-s-b* | *w-c-b* | *l-s-b* | *l-b* | *w-s-b* | *w-c-b* | *l-s-b* |
| $P_1$ | 1.2 | 0.9 | 1.3 | 1.2 | 1.2 | 1.3 | 1.5 | 1.4 | 1.1 | 1.1 | 1.4 |
| $P_2$ | 6 | 3 | 6 | 5 | 5 | 3 | 5 | 4 | 5 | 3 | 5 |
| $P_3$ | 7 | 3 | 7 | 4 | 5 | 4 | 5 | 3 | 4 | 4 | 4 |
| $P_4$ | 24 | 8 | 24 | 26 | 3 | 7 | 3 | 5 | 1.4 | 6 | 1.4 |

In the third part, the number of variables and the variable domain size are fixed ($|V| = 5$, $|D(x)| = 20$ for all variables $x$, and $|H_0| = 0$) across all instances, while problems in the same set share a specific number of hierarchy levels: $P_i$ has $i + 1$ non-required levels each with 5 constraints for $i \in \{1, 2, 3, 4\}$.

**Algorithm 2:** A Branch-and-bound CH Solver with Pruning

**bb_solv**($H$, $I_H$, $V$, $D$, $S_0$, **in out** $I_{S_0}$, $\prec_{better}$)
**begin**
  *# Any classical arc consistency algorithm*
  $D \leftarrow$ **arc_consistent**($H_0$, $D$);
  **if** $D$ *contains an empty variable domain* **then**
    **return** $S_0$;
  **else if** $D$ *contains all singleton variable domain* **then**
    **let** $\theta$ be the valuation corresponding to $D$;
    **let** $\boldsymbol{\xi}_\theta$ be the error indicator corresponding to $\theta$;
    $\boldsymbol{\xi}_\theta \leftarrow$ **cal_error_values**($H$, $\theta$, $\boldsymbol{\xi}_\theta$);
    **for** *each* $\sigma \in S_0$ **do**
      **if** $\boldsymbol{\xi}_\sigma \prec_{better} \boldsymbol{\xi}_\theta$ **then**
        $S_0 \leftarrow S_0 - \{\sigma\}$; $I_{S_0} \leftarrow I_{S_0} - \{\boldsymbol{\xi}_\sigma\}$;
      **else if** $\boldsymbol{\xi}_\theta \prec_{better} \boldsymbol{\xi}_\sigma$ **then return** $S_0$;
    $S_0 \leftarrow S_0 \cup \{\theta\}$; $I_{S_0} \leftarrow I_{S_0} \cup \{\boldsymbol{\xi}_\theta\}$;
    **return** $S_0$;
  **for** *each* $\boldsymbol{\xi}_{x=d} \in I_H$ **do**
    **if** $d \notin D(x)$ **then**
      $I_H \leftarrow I_H - \{\boldsymbol{\xi}_{x=d}\}$;
  $I_H \leftarrow$ **ch2c**($H$, $V$, $D$, $I_H$);
  **choose** variable $x \in V$ for which $|D(x)| \geq 2$;
  $W \leftarrow D(x)$;
  **for** *each* $d \in W$ **do**
    **if go**($\boldsymbol{\xi}_{x=d}$, $S_0$, $I_{S_0}$, $\prec_{better}$) **then**
      $S_0 \leftarrow$ **bb_solv**($\{H_0 \wedge x = d, H_1, \ldots, H_n\}$, $I_H$, $V$, $D$, $S_0$, $I_{S_0}$, $\prec_{better}$);
  **return** $S_0$;
**end**

Fig. 7: A Branch-and-bound CH Solver with Pruning

| | $t_g/t_c$ (Mean) | | | | $t_b/t_c$ (Mean) | | | | $t_r/t_c$ (Mean) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| CHs | *w-s-b* | *w-c-b* | *l-s-b* | *l-b* | *w-s-b* | *w-c-b* | *l-s-b* | *l-b* | *w-s-b* | *w-c-b* | *l-s-b* |
| $P_1$ | 146 | 108 | 151 | 122 | 44 | 44 | 44 | 32 | 37 | 39 | 39 |
| $P_2$ | 209 | 130 | 212 | 116 | 51 | 116 | 50 | 34 | 38 | 104 | 39 |
| $P_3$ | 232 | 168 | 219 | 50 | 42 | 121 | 44 | 21 | 31 | 113 | 29 |
| $P_4$ | 122 | 154 | 124 | 75 | 58 | 132 | 60 | 26 | 51 | 128 | 52 |

The CH-2-C algorithm incurs overhead in the branch-and-bound search. For the larger problems in $P_2$, $P_3$, and $P_4$, the extra effort paid by the CH-2-C algorithm at each search node is demonstrated worthwhile. This result is in line with the behavior of embedding classical consistency techniques in basic tree search in solving classical CSPs.

The Lua's solver relies on classical constraint propagation to enforce the semantics and the operations of the comparators via reified constraints. While the approach, based on existing technology, is clever and clean, the pruning power of reified constraints is relatively weak. On the other hand, $S_c$ executes a dedicated algorithm for maintaining CH-2-C to help pruning and solution filtering, thus attaining a higher efficiency. In particular, $S_r$ performs the worst on the *w-c-b* comparator, since the *error combining constraint* is implemented using the **IlcMax** constraint in ILOG Solver 4.4, which is again weak in propagation.

## 8 Related Work

Many efficient algorithms have been proposed to solve CHs, such as DeltaBlue [12], SkyBlue [22], DETAIL [18], Indigo [6], Generalized Local Propagation [17], and Ultraviolet [7], apply Local Propagation [24]. Besides, Cassowary and QOCA algorithms [9], adapting the Simplex algorithm [21], can also solve CHs efficiently. However, they are designed for the real number domain. We focus on finite domain CHs solving techniques; we can categorize the techniques into four different approaches.

First, the Incremental Hierarchical Constraint Solver (IHCS) [20] proposes to transform a given constraint hierarchy into a set of *best configurations* (a set of constraints). Therefore, a given CH can be transformed into a set of classical CSPs. However, it can only find *l-b* solutions using the trivial error function. The second approach is to transform CHs into ordinary constraint systems based on *reified constraint propagation* [16]. This approach can only find solutions for *global comparators* (*w-s-b*, *w-c-b*, and *l-s-b*). The third approach exploits the fact that CH is an instance of the SCSP framework [5]. Bistarelli *et al.* [4] show how a c-semiring can be constructed to model all instances of *globally-better*. In addition, only the *w-c-b* can enjoy semiring-based arc-consistency techniques [5] supported in clp(FD,S) [14]. The clp(FD,S) solver, however, limits the size of the semiring to only 32 elements, making it difficult to model any practically sized problems. The last is the refining approach used by DeltaStar [13]. It is a generic finite domain CH solver which can find solutions for arbitrary comparators in theory. However, it recomputes the solution in each recursive step causing significant overhead. Hence, it is used only as a general and theoretical framework for solution, from which efficient algorithms, such as DeltaBlue (only equality constraints) and Cassowary (a very restricted finite domain subsolver), are inspired and designed for some subset of the general problem [11].

This paper is also related to many work in soft constraint processing aiming to show how information gained through local consistency checking during preprocessing can be used to enhance branch-and-bound search using local computations as global bounds. In fact, when dealing with Constraint Hierarchies with only 2 levels, *w-s-b* and *w-l-b* correspond to weighted CSPs and *w-c-b* to fuzzy CSPs. Some work, similar to our, already appear (see for example Weighted CSPs [25], and Valued CSPs[23, 10]). The bounds computed by these works are better then ours when we restrict our computations to only 2-level, and to a specific comparator.

Our results are somewhat more general. We are able to compute bounds for CH with *any number of levels* and *without fixing a priori a comparator*. To reach better bounds we can easily fix a comparator and define a specific $approx_k(x \mapsto d)$ function. Bistarelli *et al.* [4]defined such operators for the specific case of *l-b*.

## 9 Conclusion

We formally define constraint hierarchy $k$-consistency (CH-$k$-C), based on error indicators. Incorporating a CH-2-C enforcement algorithm in a branch-and-bound algorithm, we obtain a general finite domain CH solver, which works for arbitrary comparators. Search space is pruned by utilizing the error information generated by the CH-2-C algorithm. Experiments confirm the efficiency of our research prototype, which brings us one step towards practical finite domain CH solving.

There is room for future research. First, our implementation and even the CH-2-C algorithm are hardly optimized. They have much scope for improvement. Second, we test our solver only on random problems. Experiments on more structured problems and real-life problems are needed. Third, our consistency-based and Lua's reified constraint approaches do not compete. It would be interesting to study if the two methods can be combined to produce more pruning. Fourth, the efficiency of branch-and-bound algorithms can be sensitive to variable and value orderings. It is worthwhile to investigate good ordering heuristics specific to the CH-2-C and the branch-and-bound algorithms. Fifth, the current proposal of our solver guarantees the correctness of local and global comparators. In addition, it is easy to check that our solver can support regional comparator [26], *regionally-better* comparator. The existing comparators, although rigorously and mathematically defined, might be too general for a specific real-life situation. It would be interesting to introduce new comparators that should be of particular relevance to real-life problems and applicable to our solver.

## References


[1] G.J. Badros, A. Borning, and P.J. Stuckey. The Cassowary linear arithmetic constraint solving algorithm. *ACM Transactions on Computer-Human Interaction*, 8(4):267–306, 2001.

[2] C. Bessière, E.C. Freuder, and J.C. Régin. Using inference to reduce arc consistency computation. In *Proceedings of IJCAI95*, pages 592–598, 1995.

[3] S. Bistarelli, P. Codognet, H.K.C. Hui, and J.H.M. Lee. Solving finite domain constraint hierarchies by local consistency and tree search. In *(to appear) Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, page (2 pages), 2003.

[4] S. Bistarelli, Y. Georget, and J.H.M. Lee. Capturing (fuzzy) constraint hierarchies in semiring-based constraint satisfaction. Unpublished Manuscript, http://www.sci.unich.it/∼bista/drafts/soft-fuzzyCH.pdf, 1999.

[5] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based constraint solving and optimization. *Journal of the ACM*, 44(2):201–236, 1997.

[6] A. Borning, R. Anderson, and B. Freeman-Benson. Indigo: A local propagation algorithm for inequality constraints. In *Proceedings of the 1996 ACM Symposium on User Interface Software and Technology*, pages 129–136, 1996.

[7] A. Borning and B. Freeman-Benson. Ultraviolet: A constraint satisfaction algorithm for interactive graphics. *Constraints: An International Journal*, 3(1):9–32, 1998.

[8] A. Borning, B. Freeman-Benson, and M. Wilson. Constraint hierarchies. *Lisp and Symbolic Computation*, 5(3):223–270, 1992.

[9] A. Borning, K. Marriott, P. Stuckey, and Y. Xiao. Solving linear arithmetic constraints for user interface applications. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, pages 87–96, 1997.

[10] Martin C. Cooper. Reduction operations in fuzzy or valued constraint satisfaction. *Fuzzy Sets and Systems*, 134(3):311–342, mar 2003.

[11] B. Freeman-Benson. Efficiency of DeltaStar. Private Communication, April 2002.

[12] B. Freeman-Benson, J. Maloney, and A. Borning. An incremental constraint solver. *Communications of the ACM*, 33(1):54–63, 1990.

[13] B. Freeman-Benson, M. Wilson, and A. Borning. DeltaStar: A general algorithm for incremental satisfaction of constraint hierarchies. In *The 11th Annual IEEE Phoenix Conference on Computers and Communications*, pages 561–568, 1992.

[14] Y. Georget and P. Codognet. Compiling semiring-based constraints with clp(FD,S). In *Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming*, 1998.

[15] S.A. Grant and B.M. Smith. The phase transition behavior of maintaining arc consistency. In *Proceedings of ECAI96*, pages 175–179, 1996.

[16] Martin Henz, Yun Fong Lim, Seet Chong Lua, Xiao Ping Shi, J. Paul Walser, and Roland H. C. Yap. Solving hierarchical constraints over finite domains. In *Sixth International Symposium on Artificial Intelligence and Mathematics*, Fort Lauderdale, Florida, 2000.

[17] H. Hosobe, S. Matsuoka, and A. Yonezawa. Generalized local propagation: A framework for solving constraint hierarchies. In *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming*, pages 237–251, 1996.

[18] H. Hosobe, K. Miyashita, S. Takahashi, S. Matsuoka, and A. Yonezawa. Locally simultaneous constraint satisfaction. In *Proceedings of PPCP94*, pages 51–62, 1994.

[19] A.K. Mackworth. Consistency in networks of relations. *AI Journal*, 8(1):99–118, 1977.

[20] F. Menezes, P. Barahona, and P. Codognet. An incremental hierarchical constraint solver. In *First Workshop on Principle and Practice of Constraint Processing*, 1993.

[21] J.A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7:308–313, 1965.

[22] M. Sannella. The SkyBlue constraint solver and its applications. In V.A. Saraswat and P.V. Hentenryck, editors, *Proceedings of the First Workshop on Principles and Practice of Constraint Programming*. MIT Press, 1994.

[23] Thomas Schiex. Arc consistency for soft constraints. In *Proc. 6th International Conference on Principles and Practice of Constraint Programming (CP2000)*, volume 1894, pages 411–424. Springer, 2000.

[24] G.L. Steele and G.J. Sussman. Constraints. In *APL conference proceedings part 1*, pages 208–225, 1979.

[25] Richard J. Wallace. Directed arc consistency preprocessing. In *Constraint Processing, Selected Papers*, volume 923, pages 121–137. Springer, 1995.

[26] M. Wilson and A. Borning. Hierarchical constraint logic programming. *Journal of Logic Programming*, 16:277–318, 1993.