A Foundation of Solution Methods for Constraint Hierarchies *

HIROSHI HOSOBE hosobe@nii.ac.jp
National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
SATOSHI MATSUOKA matsu@acm.org
Global Scientific Information and Computing Center, Tokyo Institute of Technology, 2-12-1 Oo-okayama, Meguro-ku, Tokyo 152-8552, Japan

Abstract. Constraint hierarchies provide a framework for soft constraints, and have been applied to areas such as artificial intelligence, logic programming, and user interfaces. In this framework, constraints are associated with hierarchical preferences or priorities called strengths, and may be relaxed if they conflict with stronger constraints. To utilize constraint hierarchies, researchers have designed and implemented various practical constraint satisfaction algorithms. Although existing algorithms can be categorized into several approaches, what kinds of algorithms are possible has been unclear from a more general viewpoint. In this paper, we propose a novel theory called generalized local propagation as a foundation of algorithms for solving constraint hierarchies. This theory formalizes a way to express algorithms as constraint scheduling, and presents theorems that support possible approaches. A benefit of this theory is that it covers algorithms using constraint hierarchy solution criteria known as global comparators, for which only a small number of algorithms have been implemented. With this theory, we provide a new classification of solution criteria based on their difficulties in constraint satisfaction. We also discuss how existing algorithms are related to our theory, which will be helpful in designing new algorithms.

Keywords: constraint hierarchies, soft constraints, constraint satisfaction, local propagation

1. Introduction

Constraint hierarchies [4, 6, 23] provide a framework for soft constraints, and have been applied to areas such as artificial intelligence [19], logic programming [21, 25], and user interfaces [4, 17, 20]. In this framework, constraints are associated with hierarchical preferences or priorities called *strengths*. Intuitively, solutions to constraint hierarchies are determined so that they will satisfy as many strong constraints as possible, leaving weaker inconsistent constraints unsatisfied. For example, the hierarchy of the constraints $\mathsf{strong}\ x = 0$ and $\mathsf{weak}\ x = 1$ yields the solution $x \leftarrow 0$. This property enables us to specify preferential or soft constraints that may be used when the set of required or hard constraints is under-constrained. Moreover, constraint hierarchies are sufficiently general to handle powerful constraints such as arithmetic equations and inequalities over the real numbers. Additionally, they allow "relaxing" constraints with the same strength by using a given solution criterion such as the least-squares method.

To utilize constraint hierarchies, researchers have designed and implemented various practical constraint satisfaction algorithms. We can categorize most existing algorithms into the following three approaches:

* An earlier version of this paper appeared as [14].

The refining method finds (all) solutions by satisfying the strongest level first and then weaker levels successively. It is mainly employed in constraint logic programming languages such as HCLP(R, $\star$) [25].

Local propagation obtains a solution by repeatedly selecting uniquely satisfiable constraints. It is used in various constraint solvers for user interfaces [9, 15, 20].

The optimization approach (approximately) computes a solution by transforming a constraint hierarchy into an optimization problem. It is adopted in recent arithmetic constraint solvers for user interfaces [7, 12, 13].

Indeed research on actual constraint satisfaction algorithms has progressed, but what kinds of algorithms are possible has been unclear from a more general viewpoint; past research has focused on how to solve hierarchies of certain kinds of constraints by adopting specific solution criteria. Desirably, we should have some general idea useful for further improving the satisfaction of constraint hierarchies.

To meet this demand, we propose a novel theory called *generalized local propagation* (GLP) as a foundation of algorithms for solving constraint hierarchies. This theory formalizes a way to express algorithms as constraint scheduling, and presents theorems that support possible approaches. Among the three major approaches mentioned above, we can actually regard the refining method and local propagation as methods for constraint scheduling. This research is the first attempt that theoretically integrates these two approaches, and also that implies the possibility of more aggressive algorithms for solving constraint hierarchies.

A benefit of this theory is that it covers algorithms using constraint hierarchy solution criteria known as global comparators, for which only a small number of algorithms have been implemented. Typically, local propagation is used to handle other criteria known as local comparators, while the refining method is employed to treat global comparators. Local comparators have been considered to be more efficiently implementable since they can be processed by "greedy" algorithms, such as local propagation, which solve one constraint at a time [1]. An important point of GLP is that it allows local propagation to handle global comparators. In other words, it expands the applicability of greedy algorithms. Actually, it gives theoretical backing to our previous *DETAIL* algorithm [15], which uses local propagation to solve constraint hierarchies with global comparators.

With this theory, we provide a classification of solution criteria based on their difficulties in constraint satisfaction. Our classification considers the applicability of the refining method and local propagation. It is new because the past classification is based on the constructive definition of solution criteria.

We also discuss how existing algorithms are related to our theory. These algorithms employ various ideas to achieve soundness, completeness, and/or efficiency, and we explain such ideas in terms of GLP. We believe that it will be helpful in designing new algorithms.

This paper is organized as follows: Section 2 describes previous research that has theoretically investigated algorithms for handling soft constraints. Section 3 provides a reformulation of constraint hierarchies to better analyze their properties. Section 4 formalizes GLP to express algorithms as constraint scheduling. Section 5 shows theoretical backgrounds of the refining method and local propagation by using GLP. Section 6 discusses the theoretical results of GLP by presenting a new classification of solution criteria and also the relationships of GLP with existing algorithms. Finally, Section 7 mentions the conclusions and future work of this research.

2. Related Work

In this section, we overview previous theoretical research on constraint hierarchies and other frameworks for soft constraints (later in Subsection 6.2, we will describe the relationships of our theory with actual algorithms for solving constraint hierarchies).

Borning et al., the originators of constraint hierarchies [4], have investigated theoretical aspects of constraint hierarchies [6, 24]. However, their theoretical analysis mainly focuses on hierarchical constraint logic programming (HCLP), and does not cover local propagation for constraint hierarchies.

Jampel constructed a certain HCLP instance that separates the HCLP scheme into compositional and non-compositional parts [16]. The method is expected to improve the efficiency of interpreters and compilers since the compositional part is efficiently implementable. However, it is unclear whether such a method is applicable to constraint scheduling in the sense of our research (we will further discuss this notion of compositionality in Subsection 6.3).

Freuder and Wallace proposed partial constraint satisfaction to handle problems that are either impossible or impractical for constraint satisfaction [10]. Theoretically, it is sufficiently general to simulate constraint hierarchies. However, their algorithms are specific ones which search for approximate solutions by "weakening" problems over finite domains.

Recently, researchers have conducted theoretical studies on constraint propagation for constraint satisfaction problems with soft constraints [2, 22]. However, their results are hard to relate with constraint hierarchies due to the large differences of the underlying frameworks.

3. Reformulation of Constraint Hierarchies

This section reformulates constraint hierarchies to facilitate the investigation of their properties. The major differences from the original formulation by Borning et al. [6] are to explicitly parameterize target hierarchies, and to replace concrete embedded functions and relations with more abstract ones.

Both of the formulations determine solutions to constraint hierarchies by comparing how well valuations (or potential solutions) satisfy constraints.

Also, they realize the hierarchical preferences of constraints by more respecting stronger constraints in the comparison. The rest of this section presents our formulation of constraint hierarchies from the bottom up.

First, we define concepts related to variables. We denote the domain of variables and the set of all variables as D and X respectively (we assume for simplicity that all variables have the same domain). To represent assignments of values to variables, we use valuations written as θ possibly with primes. Given a variable $x \in X$, $\theta(x)$ expresses the value of x, which is in D.

Definition 1. (domain; variable; valuation) Let D be the domain of variables. Let X be the set of all variables. A valuation is a mapping $\theta : X \to D$. Let Θ be the set of all valuations.

We simply consider constraints as elements of the set C. To assign semantics to constraints, we use *error functions* e that evaluate how well valuations satisfy constraints. Intuitively, $e(c, \theta)$ expresses the error of the constraint c under the valuation θ. The error being zero means that the constraint is exactly satisfied. If the error is nonzero, it must be a positive real value, and a larger value means that the constraint is worse satisfied.

Definition 2. (constraint; error function) Let C be the set of all constraints. An error function is a mapping $e : C \times \Theta \to \{0\} \cup R^+$.

Major error functions are the *predicate* and *metric* error functions. The predicate error function can be used for various kinds of constraints expressible as mathematical relations. It is defined to return 0 if a given constraint holds, and 1 otherwise.[1] By contrast, the metric error function is mainly adopted for arithmetic constraints composed of arithmetic functions and relations [18]. It expresses constraint errors as some distances. Typically, for arithmetic equality constraints, it uses the differences between the left- and right-hand sides. For example, the error of the constraint $x_1 = x_2$ may be given as follows:

$$e(\text{“}x_1 = x_2\text{”}, \theta) \equiv |\theta(x_1) - \theta(x_2)|$$

Next, we present definitions related to strengths. A strength is a non-negative integer, and a larger integer means that the strength is weaker. We write c/k to represent constraint c with strength k.

Definition 3. (strength; constraint with a strength) Let l be a constant positive integer. A strength of a constraint is an integer k such that $0 \leq k \leq l$. Let K be the set of all the strengths. Given $c \in C$ and $k \in K$, constraint c with strength k is the pair $c/k \in C \times K$.

Instead of using non-negative integers, we sometimes write strengths as symbols required, strong, medium, and weak.

Now we define constraint hierarchies as follows:

Definition 4. (constraint hierarchy) A constraint hierarchy is a finite set H of constraints with strengths. Let $\boldsymbol{H}$ be the set of all constraint hierarchies.

It should be noted that, since constraint hierarchies are ordinary sets in our formulation, a hierarchy never includes multiple instances of the same constraint. However, since the meanings of constraints are assigned by error functions, distinct constraints may work equivalently as a result. For instance, both distinct c/k and c'/k may virtually work as the same constraint $x = 0$ with strength k in a hierarchy.[2] By contrast, previous work defined constraint hierarchies by using multisets [6, 16] or vectors [23].

Next, we define *level comparators* $\overset{\cdot/k}{\leq}$ that compare how well valuations satisfy constraints at a certain level of a hierarchy. Intuitively, $\theta \overset{H/k}{\leq} \theta'$ means that θ is better than or similar to θ' in satisfying level k of H. Also, we define a relation $\overset{\cdot/k}{<}$, which indicates "better than."

Definition 5. (level comparator) Let $k \in K$. A level comparator is a ternary relation $\overset{\cdot/k}{\leq}: \boldsymbol{H} \times \Theta \times \Theta$ such that, for any $H, H' \in \boldsymbol{H}$ and $\theta, \theta', \theta'' \in \Theta$, the following conditions hold:[3]

$$\forall c \in C\,(c/k \in H \Leftrightarrow c/k \in H') \;\Rightarrow\; (\theta \overset{H/k}{\leq} \theta' \Leftrightarrow \theta \overset{H'/k}{\leq} \theta') \tag{1}$$

$$\forall c/k \in H\,(e(c,\theta) = e(c,\theta'')) \;\Rightarrow\; (\theta \overset{H/k}{\leq} \theta' \Leftrightarrow \theta'' \overset{H/k}{\leq} \theta') \tag{2}$$

$$\forall c/k \in H\,(e(c,\theta') = e(c,\theta'')) \;\Rightarrow\; (\theta \overset{H/k}{\leq} \theta' \Leftrightarrow \theta \overset{H/k}{\leq} \theta'') \tag{3}$$

$$\forall c/k \in H\,(e(c,\theta) \leq e(c,\theta')) \;\Rightarrow\; \theta \overset{H/k}{\leq} \theta' \tag{4}$$

$$\theta \overset{H/k}{<} \theta' \wedge \theta' \overset{H/k}{<} \theta'' \;\Rightarrow\; \theta \overset{H/k}{<} \theta'' \tag{5}$$

where $\theta \overset{H/k}{<} \theta'$ is defined as $\theta \overset{H/k}{\leq} \theta' \wedge \neg\, \theta' \overset{H/k}{\leq} \theta$.

Conditions (1)–(3) say that the scope of a level comparator is restricted to the inside of the designated level. Condition (4) indicates that if the error of each constraint at a level under a valuation is smaller than or equal to the one under another valuation, then the former valuation is better than or similar to the latter in satisfying the level. Condition (5) is the "transitivity" of a level comparator. The original formulation [6] includes (1)–(3) operationally and presents (5) explicitly. Also, it is consistent with (4) although it does not clarify an equivalent concept.

An instance of level comparators is the one for least-squares-better [6] (a hierarchical comparator that we will describe later). It applies the least-squares method to conflicting constraints at a level. More specifically, it compares two valuations by using the metric error function e and summing the squares of errors of constraints at the given level. Formally, it is defined as follows:

$$\theta \overset{H/k}{\leq} \theta' \Leftrightarrow \sum_{c/k \in H} e(c,\theta)^2 \leq \sum_{c'/k \in H} e(c',\theta')^2$$

It is easy to see that the definition fulfills all the conditions in Definition 5. Regarding condition (1), the level comparator will give the same results even if unrelated levels are changed. Concerning (2) and (3), it produces equivalent results as far as it obtains the same constraint errors from the associated level. With regard to (4), it evaluates a valuation no worse than another if the former always obtains either smaller or equal constraint errors. Finally, concerning (5), it exhibits transitivity because it uses the total order among the real numbers.

For convenience, we define $\overset{\cdot/k}{\geq}$ (worse than or similar to), $\overset{\cdot/k}{\sim}$ (similar to), $\overset{\cdot/k}{>}$ (worse than), and $\overset{\cdot/k}{\not\sim}$ (incomparable with) as follows: $\theta \overset{H/k}{\geq} \theta' \Leftrightarrow \theta' \overset{H/k}{\leq} \theta$; $\theta \overset{H/k}{\sim} \theta' \Leftrightarrow \theta \overset{H/k}{\leq} \theta' \wedge \theta \overset{H/k}{\geq} \theta'$; $\theta \overset{H/k}{>} \theta' \Leftrightarrow \theta \overset{H/k}{\geq} \theta' \wedge \neg\theta \overset{H/k}{\leq} \theta'$; $\theta \overset{H/k}{\not\sim} \theta' \Leftrightarrow \neg\theta \overset{H/k}{\leq} \theta' \wedge \neg\theta \overset{H/k}{\geq} \theta'$.

Next, we define *hierarchical comparators* $\dot{<}$ that compare how well valuations satisfy overall constraint hierarchies by combining level comparators. Intuitively, $\theta \overset{H}{<} \theta'$ means that θ is better than θ' in satisfying H.

Definition 6. (hierarchical comparator) A hierarchical comparator is a ternary relation $\dot{<}$: $\boldsymbol{H} \times \Theta \times \Theta$ such that, for any $H \in \boldsymbol{H}$ and $\theta, \theta' \in \Theta$,

$$\theta \overset{H}{<} \theta' \Leftrightarrow \exists k \in K\,(\forall k' \in K\,(k' < k \Rightarrow \theta \overset{H/k'}{\sim} \theta') \wedge \theta \overset{H/k}{<} \theta')$$

A hierarchical comparator is defined as a lexicographic ordering with level comparators as its components. Consequently, a level comparator has absolute priority over weaker ones. In the rest of this paper, we often refer to hierarchical comparators simply as comparators if there is no danger of misunderstanding.

For convenience, we define $\dot{>}$ (worse than), $\dot{\sim}$ (similar to), $\dot{\leq}$ (better than or similar to), $\dot{\geq}$ (worse than or similar to), and $\dot{\not\sim}$ (incomparable with) as follows: $\theta \overset{H}{>} \theta' \Leftrightarrow \theta' \overset{H}{<} \theta$; $\theta \overset{H}{\sim} \theta' \Leftrightarrow \forall k \in K\,(\theta \overset{H/k}{\sim} \theta')$; $\theta \overset{H}{\leq} \theta' \Leftrightarrow \theta \overset{H}{<} \theta' \vee \theta \overset{H}{\sim} \theta'$; $\theta \overset{H}{\geq} \theta' \Leftrightarrow \theta \overset{H}{>} \theta' \vee \theta \overset{H}{\sim} \theta'$; $\theta \overset{H}{\not\sim} \theta' \Leftrightarrow \neg\theta \overset{H}{\leq} \theta' \wedge \neg\theta \overset{H}{\geq} \theta'$.

The original formulation [6] proposed *global* and *local* comparators as major classes of hierarchical comparators. Level comparators for global comparators first compute non-negative real numbers by arithmetically integrating constraint errors, and then compare the results by using the ordinary relation $\leq$ on real numbers. Examples of global comparators are least-squares-better (LSB), weighted-sum-better (WSB), and worst-case-better (WCB). As already described, LSB uses the sum of the squares of metric constraint errors. By contrast, WSB exploits the sum of metric errors, and WCB adopts the maximum of metric errors.

Local comparators are typically used in local propagation, and are practically important. The following is their definition that has been rewritten for our constraint hierarchy formulation:

Definition 7. (local comparator) Let $k \in K$. A level comparator $\overset{\cdot/k}{\leq}$ is local if and only if for any $H \in \boldsymbol{H}$ and $\theta, \theta' \in \Theta$,

$$\theta \overset{H/k}{\leq} \theta' \Rightarrow \forall c/k \in H\,(e(c,\theta) \leq e(c,\theta')) \tag{6}$$

A hierarchical comparator is local if and only if it consists of local level comparators.

A local hierarchical comparator using the predicate error function is referred to as locally-predicate-better (LPB), and a one employing the metric function as either locally-metric-better or locally-error-better (LEB). By (4) and (6), any local level comparator results in

$$\theta \overset{H/k}{\leq} \theta' \Leftrightarrow \forall c/k \in H\,(e(c,\theta) \leq e(c,\theta'))$$

which is equivalent to the definition presented by the original formulation [6].

In addition to global and local comparators, *regional* comparators have been proposed [23]. Their level comparators are defined by separating $\overset{\cdot/k}{<}$ and $\overset{\cdot/k}{\sim}$ as follows:

$$\theta \overset{H/k}{<} \theta' \;\Leftrightarrow\; \forall c/k \in H\,(e(c,\theta) \leq e(c,\theta')) \wedge \exists c'/k \in H\,(e(c',\theta) < e(c',\theta'))$$
$$\theta \overset{H/k}{\sim} \theta' \;\Leftrightarrow\; \neg\,\theta \overset{H/k}{<} \theta' \wedge \neg\,\theta \overset{H/k}{>} \theta'$$

Regional comparators are other variations of local comparators, and are said to be more discriminative than local ones. However, to our knowledge, they have never been used in practice.

The following defines constraint hierarchy satisfiers S that represent the satisfaction of constraint hierarchies. Intuitively, given a set Θ' of valuations, $S(H,\Theta')$ is the set of valuations obtained by maximally satisfying H within Θ'. More formally, a valuation in $S(H,\Theta')$ is an element of Θ' such that there is no better valuation in Θ' in satisfying H.

Definition 8. (constraint hierarchy satisfier) Let $H \in \boldsymbol{H}$ and $\Theta' \subseteq \Theta$. A constraint hierarchy satisfier is a mapping $S : \boldsymbol{H} \times 2^{\Theta} \to 2^{\Theta}$ defined as

$$S(H,\Theta') \equiv \{\theta \in \Theta' \mid \neg\,\exists \theta' \in \Theta'\,(\theta' \overset{H}{<} \theta)\}$$

We write $S(H)$ as a shorthand of $S(H,\Theta)$.

Finally, we define solutions to constraint hierarchies. Intuitively, a solution to H is a valuation that maximally satisfies H among the set of all valuations.

Definition 9. (solution) A solution to a constraint hierarchy H is a valuation in $S(H)$.

One difference between the original formulation [6] and ours is that the original allows weights of constraints inside levels when they use global comparators, while we omitted weights for simplicity. A weight of a constraint is a positive real which scales the error of the constraint. It means that, unlike strengths, weights would always be associated with the error function. Therefore, we could easily rewrite our theory to incorporate weights.

A more significant difference is that the original formulation restricts constraints at level 0 to being required, whereas ours allows conflicting constraints at level 0. This is because our definition excluded the special treatment of level 0 for simplicity. However, as far as level 0 is not over-constrained, the resulting solutions are the same. Also, even if we added the condition for constraints at level 0 being required, we could accommodate our following results.

Particularly, if we handle constraint hierarchies with no constraints at level 0, we can easily show that our formulation is equivalent to the original. The original formulation defines solutions to hierarchies as follows:

$$\begin{aligned} S_{\mathrm{orig}}(H) &\equiv \{\theta \in S_0(H) \mid \neg\, \exists \theta' \in S_0(H)\, (\mathsf{better}(\theta', \theta, H))\} \\ \text{where} \quad S_0(H) &\equiv \{\theta \in \Theta \mid \forall c/0 \in H\, (e(c, \theta) = 0)\} \end{aligned}$$

Here the relation better is the same as our hierarchical comparator $\dot{<}$, except that better does not consider level 0. Given a hierarchy with no constraints at level 0, we have $S_0(H) = \Theta$, and therefore

$$S_{\mathrm{orig}}(H) \equiv \{\theta \in \Theta \mid \neg\, \exists \theta' \in \Theta\, (\mathsf{better}(\theta', \theta, H))\}$$

Thus $S_{\mathrm{orig}}(H)$ is equal to $S(H)$.

4. Generalized Local Propagation

Classical local propagation satisfies a constraint graph by successively solving individual constraints in an order closely associated with its graph topology. In this section, we generalize local propagation so that it can solve a set of constraints at a time and can also arbitrarily schedule such constraint sets.

To formalize this, we introduce *ordered partitions* $\langle P, \leq_P \rangle$ as follows: a partition P of a constraint hierarchy is a set generated by decomposing the hierarchy into disjoint subsets called *blocks*, and $\leq_P$ is a partial order among blocks in P.

Definition 10. (ordered partition) An ordered partition of a constraint hierarchy H is a pair $\langle P, \leq_P \rangle$ such that P is a partition of H (that is, $\bigcup_{B \in P} B = H$ and $\forall B \in P \forall B' \in P\, (B \neq B' \Rightarrow B \cap B' = \emptyset)$), and $\leq_P$ is a partial order over P. We refer to each element of P as a block, and let $B <_P B'$ denote $B \leq_P B' \wedge B \neq B'$.

Next, we define *generalized local propagation* (GLP), which obtains a set of valuations by using an ordered partition of constraints and successively satisfying blocks in a given partial order.

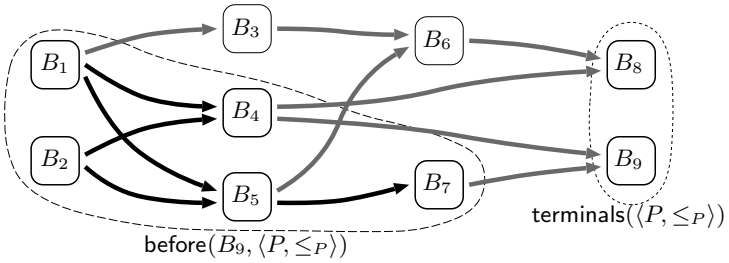

Figure 1. An ordered partition.

Definition 11. (generalized local propagation) Generalized local propagation w.r.t. an ordered partition $\langle P, \leq_P \rangle$, written as $\pi(\langle P, \leq_P \rangle)$, is defined as

$$\pi(\langle P, \leq_P \rangle) = \begin{cases} \Theta & \text{if } |P| = 0 \\ \bigcap_{B \in \mathsf{terminals}(\langle P, \leq_P \rangle)} S(B, \pi(\mathsf{before}(B, \langle P, \leq_P \rangle))) & \text{otherwise} \end{cases}$$

where terminals and before are as follows:

$$\begin{aligned} \mathsf{terminals}(\langle P, \leq_P \rangle) &= \{B' \in P \mid \neg\, \exists\, B'' \in P\, (B' <_P B'')\} \\ \mathsf{before}(B, \langle P, \leq_P \rangle) &= \langle P', \leq_{P'} \rangle \\ \text{where} \quad P' &= \{B' \in P \mid B' <_P B\} \\ \leq_{P'} &= \{\langle B', B'' \rangle \in P' \times P' \mid B' \leq_P B''\} \end{aligned}$$

Intuitively, $\mathsf{terminals}(\langle P, \leq_P \rangle)$ is the set of all blocks at terminal positions, and $\mathsf{before}(B, \langle P, \leq_P \rangle)$ is the "ordered sub-partition" of $\langle P, \leq_P \rangle$ where all blocks are before B. Since π is recursively defined, it successively solves each blocks in some order respecting $\leq_P$. This is always possible because $\leq_P$ is a partial order.

As an example, consider the ordered partition $\langle P, \leq_P \rangle$ with the blocks B_1, B_2, ..., B_9 as illustrated in Figure 1. The partial order $\leq_P$ corresponds to the reflexive transitive closure of all the arrows in Figure 1. Here $\mathsf{terminals}(\langle P, \leq_P \rangle)$ is the set $\{B_8, B_9\}$. Also, $\mathsf{before}(B_9, \langle P, \leq_P \rangle)$ is the pair consisting of the set $\{B_1, B_2, B_4, B_5, B_7\}$ and the partial order among them. Thus the satisfaction of B_9 is performed in the set of valuations obtained by applying GLP to the preceding blocks.

5. Foundation of Algorithms for Solving Constraint Hierarchies

Using generalized local propagation (GLP), this section provides theoretical backgrounds of algorithms for solving constraint hierarchies. We treat the refining method first, and local propagation later.

5.1. The Refining Method

This subsection shows a theoretical support for the refining method. To begin with, we formalize the refining method in terms of GLP. For a constraint hierarchy, we construct an ordered partition by decomposing the hierarchy into levels.

Definition 12. (the refining method) The refining method w.r.t. a constraint hierarchy H is $\pi(\langle P, \leq_P \rangle)$ such that $P = \{B_0, B_1, \ldots, B_l\}$ and $\forall k \in K\, \forall k' \in K\, (B_k \leq_P B_{k'} \Leftrightarrow k \leq k')$ where each $B_{k''} = \{c/k'' \in H\}$.

Now we show that, for any comparator, we can use the refining method to solve constraint hierarchies. It is guaranteed by the soundness of the refining method that uses an arbitrary comparator; that is, any results generated by the refining method are always solutions to the given hierarchy.

Theorem 13. Suppose that an arbitrary comparator is used. For any $H \in \boldsymbol{H}$, the refining method $\pi(\langle P, \leq_P \rangle)$ w.r.t. H satisfies $\pi(\langle P, \leq_P \rangle) \subseteq S(H)$.

Proof. See Appendix.

This theorem does not state the completeness of the refining method. In fact, we cannot obtain the completeness when using an arbitrary comparator. However, we can achieve it by adopting a restricted comparator. For this purpose, we present a class of comparators called *total* comparators, which can always compare valuations.

Definition 14. (total comparator) Let $k \in K$. A level comparator $\overset{\cdot/k}{\leq}$ is total if and only if for any $H \in \boldsymbol{H}$ and $\theta, \theta' \in \Theta$, the condition $\neg\, \theta \overset{H/k}{\not\sim} \theta'$ holds. A hierarchical comparator is total if and only if it consists of total level comparators.

Global comparators are total, since they can always compare valuations by internally comparing real numbers. Also, regional comparators are total by their definition. By contrast, local comparators are not total since sometimes they cannot compare valuations.

When exploiting total comparators, we can find all solutions by the refining method. The next theorem states the soundness and completeness of the refining method using total comparators.

Theorem 15. Suppose that a total comparator is used. For any $H \in \boldsymbol{H}$, the refining method $\pi(\langle P, \leq_P \rangle)$ w.r.t. H satisfies $\pi(\langle P, \leq_P \rangle) = S(H)$.

Proof. See Appendix.

Now we show a brief example of the refining method. Suppose that we solve a constraint hierarchy $H = \{\mathsf{strong}\ x_1 = x_2, \mathsf{medium}\ x_2 + 1 = x_3, \mathsf{weak}\ x_1 =$

$0, \mathsf{weak}\ x_3 = 3\}$ with least-squares-better. By the definition of the refining method, we let $B_0 = \{\mathsf{strong}\ x_1 = x_2\}$, $B_1 = \{\mathsf{medium}\ x_2 + 1 = x_3\}$, and $B_2 = \{\mathsf{weak}\ x_1 = 0, \mathsf{weak}\ x_3 = 3\}$. Then GLP solves H as follows: First, it obtains $S(B_0, \Theta) = \{\theta \in \Theta \mid \theta(x_1) = \theta(x_2)\}$. Next, it produces $S(B_1, S(B_0, \Theta)) = \{\theta \in \Theta \mid \theta(x_1) = \theta(x_2) \wedge \theta(x_2) + 1 = \theta(x_3)\}$. Finally, it finds $S(B_2, S(B_1, S(B_0, \Theta))) = \{\theta(x_1) = 1 \wedge \theta(x_2) = 1 \wedge \theta(x_3) = 2\}$, which can be computed by minimizing $x_1^2 + (x_3 - 3)^2$ subject to $x_1 = x_2$ and $x_2 + 1 = x_3$.

5.2. Local Propagation

The original formulation of constraint hierarchies [6] presented global and local comparators as separate classes. In this subsection, we propose a new class of comparators called *rational* comparators, which integrate most global and all local comparators. Also, we provide their fundamental properties that are useful for designing constraint satisfaction algorithms.

First, we define rational comparators. The definition is unique in that it provides how level comparators should react to the combination of constraint hierarchies.

Definition 16. (rational comparator) Let $k \in K$. A level comparator $\overset{\cdot/k}{\leq}$ is rational if and only if for any $H, H' \in \boldsymbol{H}$ such that $H \cap H' = \emptyset$ and any $\theta, \theta' \in \Theta$,

$$\theta \overset{H/k}{\leq} \theta' \wedge \theta \overset{H'/k}{<} \theta' \ \Rightarrow\ \theta \overset{H \cup H'/k}{<} \theta' \tag{7}$$

$$\theta \overset{H/k}{\not\sim} \theta' \ \Rightarrow\ \neg\, \theta \overset{H \cup H'/k}{\sim} \theta' \tag{8}$$

$$\neg\, \theta \overset{H/k}{<} \theta' \wedge \neg\, \theta \overset{H'/k}{<} \theta' \ \Rightarrow\ \neg\, \theta \overset{H \cup H'/k}{<} \theta' \tag{9}$$

A hierarchical comparator is rational if and only if it consists of rational level comparators.

It should be noted that the disjointness condition of constraint hierarchies H and H' in Definition 16 is necessary for a technical reason. It is both theoretically and practically convenient to allow multiple occurrences of the same constraint in a constraint hierarchy. However, we defined hierarchies as ordinary sets, which follows that a hierarchy can include at most one occurrence of each constraint. Instead, as mentioned in Section 3, we can simulate such multiplicity by using an error function which evaluates distinct constraints equivalently. The above disjointness condition of hierarchies is needed to guarantee distinction among constraints.

Most comparators presented by the original formulation [6] are rational. For example, least-squares-better and weighted-sum-better are rational. We can easily prove it by showing that their level comparators satisfy each condition in Definition 16. Generally, any global comparator satisfies (8), but whether it fulfills the other conditions depends on its actual definition.

Importantly, the class of rational comparators also include local comparators.

Proposition 17. Any local level comparator is rational, and any local hierarchical comparator is rational.

Proof. It is straightforward to derive all the conditions in Definition 16 from Definition 5 and Definition 7. □

A basic property of rational comparators is that any common solution to two constraint hierarchies is also a solution to their combination.

Theorem 18. Suppose that a rational comparator is used. For any $H, H' \in \boldsymbol{H}$ such that $H \cap H' = \emptyset$, the following holds:

$$S(H) \cap S(H') \subseteq S(H \cup H') \tag{10}$$

Proof. See Appendix.

An important point about this theorem is that it provides a very "weak" (or general) property that holds in many cases. It may be easily inferred from the fact that (10) is true for any two hierarchies sharing no solutions. In fact, rational comparators form a wide class including many useful comparators. It should be noted, however, that a comparator being rational is not equivalent to satisfying (10); that is, rationality is sufficient but not always necessary for (10). It suggests that there may exist irrational comparators that fulfill (10), although we have not found such a counterexample.

Among global comparators proposed by the original formulation [6], only worst-case-better (WCB) is not rational. It is because its level comparator

$$\theta \overset{H/k}{\leq} \theta' \Leftrightarrow \max_{c/k \in H} e(c, \theta) \leq \max_{c'/k \in H} e(c', \theta')$$

does not fulfill (7). Also, WCB occasionally does not satisfy (10). As an example, consider the two constraint hierarchies $H = \{\mathsf{medium}\ x_1 = 0, \mathsf{weak}\ x_1 = 4\}$ and $H' = \{\mathsf{strong}\ x_1(x_1 - 4) = 0, \mathsf{medium}\ (x_1 - 2)^2 = 0, \mathsf{weak}\ x_1 = 1\}$ over the domain $D = \{0, 1, 2, 3, 4\}$. Then both H and H' will have a unique solution $x_1 \leftarrow 0$. However, the solution to $H \cup H'$ is only $x_1 \leftarrow 4$, and thus (10) does not hold.

Now we show how GLP validates local propagation. It is more general than ordinary local propagation algorithms in the senses that it treats rational comparators as well as local ones, and also that it can solve multiple constraints at a time.

To begin with, we prove that GLP using a rational comparator respects the similarity of valuations for ordered partitions satisfying a certain condition.

Lemma 19. Suppose that a rational comparator is used. Let H be a constraint hierarchy, $\langle P, \leq_P \rangle$ an ordered partition of H, and θ an element of $\pi(\langle P, \leq_P \rangle)$. Then, for any $\theta' \in \Theta$, θ' is in $\pi(\langle P, \leq_P \rangle)$ if $\theta' \overset{H}{\sim} \theta$ and

$$\begin{aligned} &\forall B \in P\, \forall k \in K\, \forall c/k \in B\, (e(c, \theta) > 0 \\ &\quad \Rightarrow\ \forall B' \in P\, (B' <_P B \Rightarrow \forall k' \in K\, \forall c'/k' \in B'\, (k' < k))) \end{aligned} \tag{11}$$

Proof. See Appendix.

The following theorem states that such valuations are actual solutions to the constraint hierarchy:

Theorem 20. Suppose that a rational comparator is used. Let H be a constraint hierarchy, $\langle P, \leq_P \rangle$ an ordered partition of H, and θ an element of $\pi(\langle P, \leq_P \rangle)$. If (11) holds, then θ is a solution to H.

Proof. See Appendix.

Theorem 20 revealed the fact that, when using a rational comparator, GLP w.r.t. an ordered partition satisfying (11) always finds actual solutions to the original constraint hierarchy. Condition (11) means that unsatisfied constraints have only stronger constraints before them. In other words, it allows satisfiable constraints to be scheduled after weaker ones. Unlike the result of the refining method, this result of local propagation might be quite opposite to naive intuition about constraint hierarchies; it tells us that, under appropriate conditions, we are allowed to solve weak constraints before processing stronger ones. We also emphasize that it permits most global comparators as well as local ones.

Now we illustrate an example of local propagation using a global comparator. Consider a constraint hierarchy $H = \{\mathsf{strong}\ x_1 = 0, \mathsf{strong}\ x_2 + x_3 = x_4, \mathsf{medium}\ x_1 = x_2, \mathsf{medium}\ x_2 = 2, \mathsf{weak}\ x_3 = 1\}$ with least-squares-better. We use an ordered partition whose blocks are $B_1 = \{\mathsf{strong}\ x_1 = 0\}$, $B_2 = \{\mathsf{medium}\ x_1 = x_2, \mathsf{medium}\ x_2 = 2\}$, $B_3 = \{\mathsf{weak}\ x_3 = 1\}$, and $B_4 = \{\mathsf{strong}\ x_2 + x_3 = x_4\}$, and also whose partial order is composed of $B_1 <_P B_2$, $B_2 <_P B_4$, and $B_3 <_P B_4$. Then GLP processes it as follows: First, it obtains $S(B_1, \Theta) = \{\theta \in \Theta \mid \theta(x_1) = 0\}$. Second, it yields $S(B_2, S(B_1, \Theta)) = \{\theta \in \Theta \mid \theta(x_1) = 0 \land \theta(x_2) = 1\}$. Next, it acquires $S(B_3, \Theta) = \{\theta \in \Theta \mid \theta(x_3) = 1\}$. Finally, it generates $S(B_4, S(B_2, S(B_1, \Theta)) \cap S(B_3, \Theta)) = \{\theta(x_1) = 0 \land \theta(x_2) = 1 \land \theta(x_3) = 1 \land \theta(x_4) = 2\}$. Note that both of the medium constraints are relaxed while the weak constraint is exactly satisfied. Nevertheless, the solution is correct because the ordered partition satisfies the sufficient condition in Theorem 20; that is, the relaxed medium constraints have only a stronger constraint $\mathsf{strong}\ x_1 = 0$ before them. Also, note that we may solve the medium and weak constraints before $\mathsf{strong}\ x_2 + x_3 = x_4$ since the strong constraint is exactly satisfiable.

Most local propagation algorithms handle only locally-predicate-better. In the next theorem, we show that, for local comparators, we can use a weaker sufficient condition than the one for rational comparators:

Theorem 21. Suppose that a local comparator is used. Let H be a constraint hierarchy, $\langle P, \leq_P \rangle$ an ordered partition of H, and θ an element of $\pi(\langle P, \leq_P \rangle)$. Then θ is a solution to H if

$$\begin{aligned} &\forall B \in P\, \forall k \in K\, \forall c/k \in B\, (e(c, \theta) > 0 \\ &\quad \Rightarrow\ \forall B' \in P\, (B' <_P B \Rightarrow \forall k' \in K\, \forall c'/k' \in B'\, (k' \leq k))) \end{aligned} \tag{12}$$

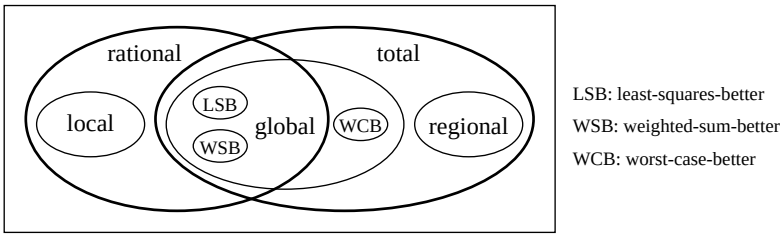

Figure 2. Classification of comparators.

Proof. Similar to the proofs of Lemma 19 and Theorem 20. □

Theorem 21 points out that GLP adopting local comparators can obtain solutions to constraint hierarchies in a weaker condition than GLP using ordinary rational ones. The difference of (12) from (11) is the existence of an equality in the comparison of strengths. It enables GLP adopting local comparators to solve constraints with equal strengths in an arbitrary order.

6. Discussion

We discuss the theoretical results of generalized local propagation (GLP) in this section.

6.1. Classification of Comparators

Our classification of comparators added two new classes, total and rational comparators. As illustrated in Figure 2, total comparators include all global and regional ones, and rational ones include most global and all local ones.

The key point about our classification is that it is based on difficulties in constraint satisfaction. Total comparators achieve both the completeness and soundness in the refining method, whereas non-total ones guarantee only the soundness. Rational comparators enable local propagation, and local ones allow more aggressive local propagation.

The most significant contribution is the result related to rational but non-local comparators such as least-squares-better and weighted-sum-better. Since these comparators are practically important [7], their potential for efficient constraint satisfaction, which we disclosed, should be informative to algorithm designers.

6.2. Relationships with Existing Algorithms

We can actually relate GLP with existing algorithms for solving constraint hierarchies. The points are which comparator an algorithm uses and how it schedules constraints.

As shown in the previous section, we can solve various constraint hierarchies using the refining method. For example, ThingLab [4] solves constraint

hierarchies by processing the levels strongest to weakest, and finds locally-error-better (LEB) solutions if there are no cyclic dependencies of constraints, and otherwise least-squares-better ones by using the relaxation technique. The Orange algorithm [8, 9, 23] obtains either weighted-sum-better (WSB) or worst-case-better (WCB) solutions to hierarchies of linear equality and inequality constraints by performing the simplex method at each level.

An interesting refining algorithm is DeltaStar [8, 23]. Basically, it satisfies a constraint hierarchy with WSB, WCB, or LEB by successively applying a "flat" solver to each level. To ensure the completeness for LEB, it maintains a family of sets of solutions for each level, which can be viewed as using multiple ordered partitions in GLP. To realize incremental constraint satisfaction, it memorizes a valuation set for each level, and further decomposes levels of hierarchies into disjoint sets.

Most existing local propagation algorithms solve hierarchies of multi-way dataflow (or functional) constraints with locally-predicate-better. Blue [17] is the first algorithm in this category. Given a constraint hierarchy, it constructs a dataflow graph by successively selecting one of the strongest uniquely satisfiable constraints. We can regard such a graph as an ordered partition in Theorem 21.

The DeltaBlue algorithm [9, 17] is an incremental version of Blue. It incrementally updates dataflow graphs by exploiting a special data structure "walkabout strength." The algorithm is guaranteed by the blocked constraint lemma [9], which we can consider as a specialization of Theorem 21.

DETAIL [15] is a local propagation algorithm that approximately handles global comparators such as least-squares-better. It decomposes a constraint hierarchy into disjoint components by adopting walkabout strengths, which we can view as an ordered partition in Theorem 20.

We can further restrict Theorem 21 on GLP for local comparators so that it processes constraints strongest to weakest; that is, it handles the strongest level first and weaker ones later, and also treats constraints with equal strengths in an arbitrary order. This strategy has been actually employed in certain algorithms [3, 5, 11, 12, 26].

An interesting example is the Indigo algorithm [3], which adopts interval propagation to solve hierarchies of equality and inequality constraints with LEB. It processes constraints strongest to weakest, and tries to maximally satisfy each constraint by tightening and propagating bounds on variables. We can regard this process as GLP for local comparators.

Another example is the Ultraviolet algorithm [5], which computes an LEB solution of a given hierarchy by first partitioning it into disjoint components called regions, and then invoking appropriate "subsolvers" such as Blue and Indigo. In selecting subsolvers for regions, it considers kinds of variables and constraints as well as the existence of cycles of constraints. Then it further separates each region according to strengths, and solves resulting sub-regions strongest to weakest. Thus we can view Ultraviolet as GLP using a local comparator.

6.3. Other Issues

A compositional theory of constraint hierarchies [16] was proposed for the efficient implementation of hierarchical constraint logic programming languages. It aims at efficiently solving the combination of hierarchies by composing their solutions which have been computed separately. Therefore, the compositional theory and ours share the same motivation in a sense. However, its approach is quite technically different from ours. It first introduces a concept called BCH (which stands for "Bags for the Composition of Hierarchies") to provide constraint problems which are compositional but lack a hierarchical order of preferential levels. Next, to realize the hierarchy, it integrates a non-compositional operation called FGH ("Filters, Guards and Hierarchies") in a similar way to the refining method. Also, the compositional theory does not present the analysis of comparators other than locally-predicate-better.

Many existing local propagation algorithms [9, 15, 20] realize incrementality in constraint satisfaction, which is not treated by our theory. These algorithms efficiently solve modified constraint hierarchies by incrementally updating dataflow graphs for previous hierarchies. It means that incrementality needs to handle two slightly different hierarchies and consider their difference in constraint scheduling, which we did not cover in this research. We feel that treating incrementality will require further knowledge about properties of specific kinds of constraints.

7. Conclusions and Future Work

In this paper, we proposed generalized local propagation (GLP) as a foundation of algorithms for solving constraint hierarchies. It formalized a way to express algorithms as constraint scheduling, and presented theorems that support the refining method and local propagation. Also, we provided a new classification of comparators based on their difficulties in constraint satisfaction, and discussed how existing algorithms are related to GLP.

Our future work includes treating the "dynamic" update of ordered partitions as well as a single "static" partition used by current GLP. This enhancement will be useful to discuss, for example, the SkyBlue algorithm [20], which uses backtracking to search for solutions. Another important future direction is to consider the optimization approach. From our recent experience in an optimization-based nonlinear constraint solver Chorus [13], we feel that it is difficult to improve accuracy as well as efficiency in the satisfaction of nonlinear constraint hierarchies. We believe that it will be effective to integrate constraint scheduling with the optimization approach.

Appendix

Proof of Theorem 13. Assume for contradiction that there exists some θ that is in $\pi(\langle P, \leq_P \rangle)$ but not in $S(H)$. Since $\theta \notin S(H)$, for some $\theta' \in \Theta$,

$\theta' \overset{H}{<} \theta$ holds; that is, for some $k \in K$, we have $\forall k' \in K\,(k' < k \Rightarrow \theta' \overset{H/k'}{\sim} \theta) \wedge \theta' \overset{H/k}{<} \theta$. By the definition of P and (1) in Definition 5, we have $\forall k' \in K\,(k' < k \Rightarrow \theta' \overset{B_{k'}/k'}{\sim} \theta) \wedge \theta' \overset{B_k/k}{<} \theta$. As θ must be in $\pi(\mathsf{before}(B_k, \langle P, \leq_P\rangle))$ and $\forall k' \in K\,(k' < k \Rightarrow \theta' \overset{B_{k'}/k'}{\sim} \theta)$, $\theta' \in \pi(\mathsf{before}(B_k, \langle P, \leq_P\rangle))$ holds by Definition 11. Then $\theta' \overset{B_k/k}{<} \theta$ implies $\theta \notin S(B_k, \pi(\mathsf{before}(B_k, \langle P, \leq_P\rangle)))$, which is a contradiction to $\theta \in \pi(\langle P, \leq_P\rangle)$. □

Proof of Theorem 15. By Theorem 13, it suffices to show $\pi(\langle P, \leq_P\rangle) \supseteq S(H)$. Assume for contradiction that there exists some θ that is in $S(H)$ but not in $\pi(\langle P, \leq_P\rangle)$. Since $\theta \notin \pi(\langle P, \leq_P\rangle)$, for some $k \in K$, we have $\theta \in \pi(\mathsf{before}(B_k, \langle P, \leq_P\rangle))$ and $\theta \notin S(B_k, \pi(\mathsf{before}(B_k, \langle P, \leq_P\rangle)))$; that is, for some $\theta' \in \pi(\mathsf{before}(B_k, \langle P, \leq_P\rangle))$, $\theta' \overset{B_k/k}{<} \theta$ holds. Also, as $\dot{<}$ is total, for any $k' \in K$ such that $k' < k$, we have $\theta' \overset{B_{k'}/k'}{<} \theta \vee \theta' \overset{B_{k'}/k'}{\sim} \theta \vee \theta' \overset{B_{k'}/k'}{>} \theta$. Hence we have the following three cases:

1. Case $\exists k' \in K\,(k' < k \wedge \forall k'' \in K\,(k'' < k' \Rightarrow \theta' \overset{B_{k''}/k''}{\sim} \theta) \wedge \theta' \overset{B_{k'}/k'}{<} \theta)$: By the definition of P, we have $\exists k' \in K\,(k' < k \wedge \forall k'' \in K\,(k'' < k' \Rightarrow \theta' \overset{H/k''}{\sim} \theta) \wedge \theta' \overset{H/k'}{<} \theta)$, and then $\theta' \overset{H}{<} \theta$ holds, which is a contradiction to $\theta \in S(H)$.

2. Case $\forall k' \in K\,(k' < k \Rightarrow \theta' \overset{B_{k'}/k'}{\sim} \theta)$: By the definition of P, we have $\forall k' \in K\,(k' < k \Rightarrow \theta' \overset{H/k'}{\sim} \theta) \wedge \theta' \overset{H/k}{<} \theta)$, and then $\theta' \overset{H}{<} \theta$ holds, which is a contradiction to $\theta \in S(H)$.

3. Case $\exists k' \in K\,(k' < k \wedge \forall k'' \in K\,(k'' < k' \Rightarrow \theta' \overset{B_{k''}/k''}{\sim} \theta) \wedge \theta' \overset{B_{k'}/k'}{>} \theta)$: Since θ must be in $\pi(\mathsf{before}(B_{k'}, \langle P, \leq_P\rangle))$, $\theta' \notin S(B_{k'}, \pi(\mathsf{before}(B_{k'}, \langle P, \leq_P\rangle)))$ holds, which is a contradiction to $\theta' \in \pi(\mathsf{before}(B_k, \langle P, \leq_P\rangle))$.

All the three cases resulted in contradictions. □

Proof of Theorem 18. Assume for contradiction that there exists some θ that is in both $S(H)$ and $S(H')$ but not in $S(H \cup H')$. Then, for some $\theta' \in \Theta$, $\theta' \overset{H\cup H'}{<} \theta$ holds; that is, for some $k \in K$, we have $\forall k' \in K\,(k' < k \Rightarrow \theta' \overset{H\cup H'/k'}{\sim} \theta) \wedge \theta' \overset{H\cup H'/k}{<} \theta$. By (7) and (8), $\theta' \overset{H\cup H'/k'}{\sim} \theta$ implies $(\theta' \overset{H/k'}{<} \theta \wedge \theta' \overset{H'/k'}{>} \theta) \vee (\theta' \overset{H/k'}{\sim} \theta \wedge \theta' \overset{H'/k'}{\sim} \theta) \vee (\theta' \overset{H/k'}{>} \theta \wedge \theta' \overset{H'/k'}{<} \theta)$, and by (9), $\theta' \overset{H\cup H'/k}{<} \theta$ implies $\theta' \overset{H/k}{<} \theta \vee \theta' \overset{H'/k}{<} \theta$. Hence we have the following two cases:

1. Case $\exists k' \in K\,(k' \leq k \wedge \forall k'' \in K\,(k'' < k' \Rightarrow \theta' \overset{H/k''}{\sim} \theta \wedge \theta' \overset{H'/k''}{\sim} \theta) \wedge \theta' \overset{H/k'}{<} \theta)$: Then $\theta' \overset{H}{<} \theta$ holds, which is a contradiction to $\theta \in S(H)$.

2. Case $\exists k' \in K\,(k' \leq k \wedge \forall k'' \in K\,(k'' < k' \Rightarrow \theta' \overset{H/k''}{\sim} \theta \wedge \theta' \overset{H'/k''}{\sim} \theta) \wedge \theta' \overset{H'/k'}{<} \theta)$: Then $\theta' \overset{H'}{<} \theta$ holds, which is a contradiction to $\theta \in S(H')$.

Both of the two cases resulted in contradictions. □

Proof of Lemma 19. Assume for contradiction that there exists some $\theta' \in \Theta$ such that $\theta' \overset{H}{\sim} \theta$ and $\theta' \notin \pi(\langle P, \leq_P \rangle)$. Then, by Definition 11, it is necessary that, for some $B_1 \in P$, θ' is in $\pi(\mathsf{before}(B_1, \langle P, \leq_P \rangle))$ but not in $S(B_1, \pi(\mathsf{before}(B_1, \langle P, \leq_P \rangle)))$. Since we have $\theta' \overset{H}{\sim} \theta$ and S is rational, $\theta \overset{B_1}{\not\sim} \theta'$ does not hold. Therefore, by the definition of θ', $\theta \overset{B_1}{<} \theta'$ must hold; that is, for some $k_1 \in K$, we have $\forall k \in K\,(k < k_1 \Rightarrow \theta \overset{B_1/k}{\sim} \theta') \wedge \theta \overset{B_1/k_1}{<} \theta'$. This implies that, for some $B \in P$, $\theta \overset{B/k_1}{>} \theta'$ holds. As θ must be in $S(B, \pi(\mathsf{before}(B, \langle P, \leq_P \rangle)))$, we have the following two cases:

1. Case $\theta' \in \pi(\mathsf{before}(B, \langle P, \leq_P \rangle)) \wedge \theta' \notin S(B, \pi(\mathsf{before}(B, \langle P, \leq_P \rangle)))$: Since $\theta \overset{B/k_1}{>} \theta'$ holds, there must exist some $k_2 \in K$ such that $k_2 < k_1$ and $\theta \overset{B/k_2}{<} \theta'$.

2. Case $\theta' \notin \pi(\mathsf{before}(B, \langle P, \leq_P \rangle))$: Then, for some $B' \in P$ such that $B' <_P B$, θ' is in $\pi(\mathsf{before}(B', \langle P, \leq_P \rangle))$ but not in $S(B', \pi(\mathsf{before}(B', \langle P, \leq_P \rangle)))$. Since, by (4) in Definition 5, $\theta \overset{B/k_1}{>} \theta'$ implies $\exists c/k_1 \in B\,(e(c, \theta) > 0)$, and also since (11) holds, B' contains only stronger constraints than k_1. Therefore, there exists some $k_2 \in K$ such that $k_2 < k_1$ and $\theta \overset{B'/k_2}{<} \theta'$.

In both of the two cases, $\theta \overset{B_1/k_1}{<} \theta'$ resulted in that there exist some $k_2 \in K$ and $B_2 \in P$ such that $k_2 < k_1$ and $\theta \overset{B_2/k_2}{<} \theta'$. Therefore, it causes an infinite sequence $k_1, k_2, \ldots$ such that $k_i > k_{i+1}$, which is a contradiction to that each k_i is a non-negative integer. □

Proof of Theorem 20. By induction on the size of P:

Induction base. If $|P| = 0$, the proposition holds.

Induction step. Assume that, if $|P| < n$, the proposition holds. Now let $|P| = n$. For any $B \in \mathsf{terminals}(\langle P, \leq_P \rangle)$, θ must be in $S(B, \pi(\mathsf{before}(B, \langle P, \leq_P \rangle)))$. Therefore, by the induction hypothesis, θ is in $S(H_B)$, where H_B is the union of blocks of $\mathsf{before}(B, \langle P, \leq_P \rangle)$. Now we assume for contradiction that there exists some $\theta' \in \Theta$ such that $\theta' \overset{H_B \cup B}{<} \theta$; that is, for some $k \in K$, we have $\forall k' \in K\,(k' < k \Rightarrow \theta' \overset{H_B \cup B/k'}{\sim} \theta) \wedge \theta' \overset{H_B \cup B/k}{<} \theta$. Hence we have the following two cases:

1. Case $\exists k' \in K\,(k' \leq k \wedge \forall k'' \in K\,(k'' < k' \Rightarrow \theta' \overset{H_B/k''}{\sim} \theta \wedge \theta' \overset{B/k''}{\sim} \theta) \wedge \theta' \overset{H_B/k'}{<} \theta)$: Then $\theta' \overset{H_B}{<} \theta$ holds, which is a contradiction to $\theta \in S(H_B)$.

2. Case $\exists k' \in K\,(k' \leq k \wedge \forall k'' \in K\,(k'' < k' \Rightarrow \theta' \overset{H_B/k''}{\sim} \theta \wedge \theta' \overset{B/k''}{\sim} \theta) \wedge \theta' \overset{B/k'}{<} \theta)$: Then, by (4) in Definition 5, for some $c/k' \in B$, $e(c, \theta) > 0$ must hold. By (11), H_B contains only stronger constraints than k'. Therefore, $\theta' \overset{H_B}{\sim} \theta$ holds. By Lemma 19, θ' is also in $\pi(\mathsf{before}(B, \langle P, \leq_P \rangle))$. As $\theta' \overset{B}{<} \theta$, it is a contradiction to $\theta \in S(B, \pi(\mathsf{before}(B, \langle P, \leq_P \rangle)))$.

Both of the two cases resulted in contradictions. Therefore, there never exists such θ'; that is, θ is in $S(H_B \cup B)$. Since the comparator is rational, θ is in $S(H)$ by Theorem 18. □

Acknowledgements

We would like to thank the referees and editors for their helpful comments and suggestions. We would like to express our deep gratitude to Akinori Yonezawa for his generous and continuous support. We are also grateful to the members of the TRIP user interface research group at the University of Tokyo and the Tokyo Institute of Technology for their patient discussions.

Notes

[1] Practically, it is sometimes necessary to consider a certain degree of computation errors to judge whether a constraint is satisfied.

[2] A constraint hierarchy with an overlap of constraints in such a sense may yield solutions different from ones to a hierarchy without the overlap. It is because the overlapping constraints will more influence the decision of solutions.

[3] When we write $\forall c/k \in H$, we mean that the universal quantifier $\forall$ is associated only with c. In other words, k is either free or quantified by another preceding one.

References

1. Badros, G. J. and A. Borning: 1998, 'The Cassowary Linear Arithmetic Constraint Solving Algorithm: Interface and Implementation'. Technical Report 98-06-04, Dept. of Computer Science and Engineering, University of Washington.
2. Bistarelli, S., R. Gennari, and F. Rossi: 2000, 'Constraint Propagation for Soft Constraints: Generalization and Termination Conditions'. In: *Principles and Practice of Constraint Programming—CP2000*, Vol. 1894 of *LNCS*. Springer, pp. 83–97.
3. Borning, A., R. Anderson, and B. Freeman-Benson: 1996, 'Indigo: A Local Propagation Algorithm for Inequality Constraints'. In: *Proc. ACM UIST*. pp. 129–136.
4. Borning, A., R. Duisberg, B. Freeman-Benson, A. Kramer, and M. Woolf: 1987, 'Constraint Hierarchies'. In: *Proc. ACM OOPSLA*. pp. 48–60.
5. Borning, A. and B. Freeman-Benson: 1998, 'Ultraviolet: A Constraint Satisfaction Algorithm for Interactive Graphics'. *Constraints J.* **3**(1), 9–32.

6. Borning, A., B. Freeman-Benson, and M. Wilson: 1992, 'Constraint Hierarchies'. *Lisp and Symbolic Comput.* **5**(3), 223–270.
7. Borning, A., K. Marriott, P. Stuckey, and Y. Xiao: 1997, 'Solving Linear Arithmetic Constraints for User Interface Applications'. In: *Proc. ACM UIST*. pp. 87–96.
8. Freeman-Benson, B., M. Wilson, and A. Borning: 1992, 'DeltaStar: A General Algorithm for Incremental Satisfaction of Constraint Hierarchies'. In: *Proc. IEEE IPCCC*. pp. 561–568.
9. Freeman-Benson, B. N., J. Maloney, and A. Borning: 1990, 'An Incremental Constraint Solver'. *Commun. ACM* **33**(1), 54–63.
10. Freuder, E. C. and R. J. Wallace: 1992, 'Partial Constraint Satisfaction'. *Artif. Intell.* **58**, 21–70.
11. Harvey, W., P. Stuckey, and A. Borning: 1997, 'Compiling Constraint Solving using Projection'. In: *Principles and Practice of Constraint Programming—CP97*, Vol. 1330 of *LNCS*. Springer, pp. 491–505.
12. Hosobe, H.: 2000, 'A Scalable Linear Constraint Solver for User Interface Construction'. In: *Principles and Practice of Constraint Programming—CP2000*, Vol. 1894 of *LNCS*. Springer, pp. 218–232.
13. Hosobe, H.: 2001, 'A Modular Geometric Constraint Solver for User Interface Applications'. In: *Proc. ACM UIST*. pp. 91–100.
14. Hosobe, H., S. Matsuoka, and A. Yonezawa: 1996, 'Generalized Local Propagation: A Framework for Solving Constraint Hierarchies'. In: *Principles and Practice of Constraint Programming—CP96*, Vol. 1118 of *LNCS*. Springer, pp. 237–251.
15. Hosobe, H., K. Miyashita, S. Takahashi, S. Matsuoka, and A. Yonezawa: 1994, 'Locally Simultaneous Constraint Satisfaction'. In: *Principles and Practice of Constraint Programming—PPCP'94*, Vol. 874 of *LNCS*. Springer, pp. 51–62.
16. Jampel, M.: 1996, 'A Compositional Theory of Constraint Hierarchies (Operational Semantics)'. In: *Over-Constrained Systems*, Vol. 1106 of *LNCS*. Springer, pp. 189–206.
17. Maloney, J. H., A. Borning, and B. N. Freeman-Benson: 1989, 'Constraint Technology for User-Interface Construction in ThingLab II'. In: *Proc. ACM OOPSLA*. pp. 381–388.
18. Marriott, K. and P. J. Stuckey: 1998, *Programming with Constraints: An Introduction*. MIT Press.
19. Ryu, Y. U.: 1999, 'A Hierarchical Constraint Satisfaction Approach to Product Selection for Electronic Shopping Support'. *IEEE Trans. Syst., Man, Cybern. A* **29**(6), 525–532.
20. Sannella, M.: 1994, 'SkyBlue: A Multi-Way Local Propagation Constraint Solver for User Interface Construction'. In: *Proc. ACM UIST*. pp. 137–146.
21. Satoh, K. and A. Aiba: 1991, 'Computing Soft Constraints by Hierarchical Constraint Logic Programming'. Technical Report TR-610, ICOT, Japan.
22. Schiex, T.: 2000, 'Arc Consistency for Soft Constraints'. In: *Principles and Practice of Constraint Programming—CP2000*, Vol. 1894 of *LNCS*. Springer, pp. 411–424.
23. Wilson, M.: 1993, 'Hierarchical Constraint Logic Programming (Ph.D. Dissertation)'. Technical Report 93-05-01, Dept. of Computer Science and Engineering, University of Washington.
24. Wilson, M. and A. Borning: 1989, 'Extending Hierarchical Constraint Logic Programming: Nonmonotonicity and Inter-Hierarchy Comparison'. In: *Proc. North American Conf. on Logic Programming*. pp. 3–19.
25. Wilson, M. and A. Borning: 1993, 'Hierarchical Constraint Logic Programming'. *J. Logic Prog.* **16**(3&4), 277–319.
26. Wolf, A.: 1996, 'Transforming Ordered Constraint Hierarchies into Ordinary Constraint Systems'. In: *Over-Constrained Systems*, Vol. 1106 of *LNCS*. Springer, pp. 171–187.