An Interval Constraint Branching Scheme for Lattice Domains

Antonio J. Fernández*
Depto. de Lenguajes y Ciencias de la Computación
E.T.S.I.I., 29071 Teatinos
University of Málaga
Málaga, Spain
afdez@lcc.uma.es

Patricia M. Hill
School of Computing
Leeds, LS2 9JT
University of Leeds
Leeds, England
hill@comp.leeds.ac.uk

Abstract

This paper presents a parameterized schema for interval constraint branching that (with suitable instantiations of the parameters) can solve interval constraint satisfaction problems (CSPs) that are defined on any set of computation domains (finite or infinite) that are lattices. A formal specification of the schema and a number of interesting properties, satisfied by any instance of the schema, are presented. It is also shown that the operational procedures of many constraint systems (including cooperative systems) are instances of this branching schema.

Keywords: interval constraints, constraint solving, propagation, branching.

1 Introduction

In [6] we described a generic interval constraint propagation schema to solve CSPs (i.e., a set of interval constraints defined on a set of lattice structure computation domains). However, although our propagation schema guaranteed finding a *most general solution* to the constraint store representing a CSP, it was not complete in the sense that it may not determine which values in the domains (i.e., intervals) of the constrained variables are the correct answers to the problem.

This paper proposes a branching schema that is complementary to the constraint propagation schema described in [6]. The combination of these two schemas forms a complete[1] interval constraint solving framework that can be used on any set of domains which have lattice structure, independently of their nature and, in particular, their cardinality. As a consequence it can be used for most existing constraint domains (finite or continuous) and, as for the framework described in [6], is also applicable to multiple domains and cooperative systems.

We also describe here some interesting properties that are satisfied by any instance of the branching schema.

2 Preliminaries and Notation

In the following we introduce some concepts and notations already described in [6]. If C is a set, then $\#C$ denotes its cardinality, $\wp(C)$ its power set and $\wp_f(C)$ the set of all the finite subsets of C.

The domain on which the values are actually computed, is called a *computation domain*. Throughout the paper, we let $\mathcal{L}$ denote a (possibly infinite) set of computation domains, with lattice structure, containing at least one element L. If it exists, $\bot_L$ (resp. $\top_L$) denotes *the bottom element* (resp. *the top element*) of L. With each computation domain

*This author was partially supported by Spanish MCyT under contracts TIC2002-04498-C05-02 and TIN2004-7943-C04-01.

[1] 'Complete' in the sense that the correctness and completeness of the branching schema can be guaranteed.

$L \in \mathcal{L}$, we associate a set of variable symbols V_L that is disjoint from $V_{L'}$ for any $L' \in \mathcal{L}$ distinct of L. We define $\mathcal{V}_\mathcal{L} = \cup\{V_L | L \in \mathcal{L}\}$. It is assumed (without loss of generality) that all $L \in \mathcal{L}$ are lifted lattices[2]. $X \in \wp_f(\mathcal{V}_\mathcal{L})$ denotes the set of constrained variables.

To allow for continuous and infinite domains, any underlying computation domain L is first replaced by two extended forms of the domain, a left and a right *bounded computation domain*. For it, we defined open and closed bounds of the intervals; we first defined the *bracket domain* B as a lattice containing just ')' and ']' with ordering ')' $\prec_B$ ']'. We let '}' denote any element of B. Then we constructed the (right) *simple bounded computation domain* (for L) to be the lattice resulting from the lexicographic product (L, B) and is denoted L^s. Throughout the paper, an element t=(a,'}') $\in L^s$ will be denoted indistinctly as 'a}' or $\mathrm{a}_\}$. The *mirror (of L^s)* (also called the left *simple bounded computation domain*) is the lexicographic product $(\hat{L}, B)$ (where $\hat{L}$ is the dual lattice of L) and is denoted by $\overline{L^s}$. The *mirror* of an element t=(a,'}') $\in L^s$ is the element $(\hat{\mathrm{a}}, \text{'}\}\text{'}) \in \overline{L^s}$ and is denoted indistinctly as $\overline{t}$, '{a', $\hat{\mathrm{a}}_\}$ or simply $\overline{\mathrm{a}_\}}$ as it is evident that if $\overline{t} = \hat{\mathrm{a}}_\}$ then $t = \mathrm{a}_\}$.

To enable user defined propagation and constraint cooperation we also extended the simple bounded computation domain (i.e., L^s) to include an additional construct called an *indexical* (i.e., functions that allowed to propagate the bounds of the interval associated to constrained variables) to form a new domain called the *bounded computation domain* L^b. Then, the *interval domain* R^b_L over L is the direct product $\langle \overline{L^b}, L^b \rangle$ whereas the *simple interval domain* R^s_L over L is the direct product $\langle \overline{L^s}, L^s \rangle$.

An element $r \in R^b_L$ is called a *range*. If $r \in R^s_L$, then we say it is *simple*. A simple range $r = \langle \overline{s}, t \rangle$ (also denoted indistinctly as $\overline{s}, t$) is *consistent* if[3] $s \preceq_{L^s} t$ and, if $s = a_)$ and, for some $b \in B$, $t = a'_b$, then $a \neq a'$.

Let $x \in V_L$. Then $x \sqsubseteq r$ is called an *interval constraint* for L with *constrained variable* x if $r \in R^b_L$. Also, $x \sqsubseteq r$ is simple (consistent) is r is simple (consistent), and non-simple (inconsistent) otherwise. If $t \in L$, then $x = t$ is a shorthand for $x \sqsubseteq [\mathbf{t}, \mathbf{t}]$. The *interval constraints domain over X for L* is the set of all interval constraints for L with constrained variables in X and is denoted by $\mathcal{C}^X_L$. The union

$$\mathcal{C}^X \stackrel{\text{def}}{=} \bigcup \{\mathcal{C}^X_L \mid L \in \mathcal{L}\}$$

is called the *interval constraint domain over X for $\mathcal{L}$*. The ordering for $\mathcal{C}^X$ is inherited from the ordering in R^s_L. We define $c_1 \preceq_{\mathcal{C}^X} c_2$ if and only if, for some $L \in \mathcal{L}$, $c_1 = x \sqsubseteq r_1, c_2 = x \sqsubseteq r_2 \in \mathcal{C}^X_L$ and $r_1 \preceq_{R^s_L} r_2$.

If $S \in \wp_f(\mathcal{C}^X)$, then S is a *constraint store* for X. If S contains only simple constraints, then it is *simple*. If S is simple, then it is *consistent* if all its constraints are consistent. The set of all simple constraint stores for X is denoted by $\mathcal{S}^X$. A constraint store S is *stable* if there is exactly one simple constraint for each $x \in X$ in S. The set of all simple stable constraint stores for X is denoted by $\mathcal{SS}^X$.

Let $S, S' \in \mathcal{SS}^X$ where c_x, c'_x denote the (simple) constraints for $x \in X$ in S and S', respectively. Then $S \preceq S'$ if and only if, for each $x \in X$, $c_x \preceq c'_x$. Let $\top_{\mathcal{SS}^X}$ be the set $\{x \sqsubseteq \bot_{R^s_L} \mid x \in X \cap V_L, L \in \mathcal{L}\}$. Then, with these definitions, $\mathcal{SS}^X$ forms a lattice.

We defined a generic concept of constraint precision. Let $\mathcal{CC}^X_L$ be the set of all consistent (and thus simple) interval constraints for L with constrained variables in X, $x \in X \cap V_L$ for any $L \in \mathcal{L}$ and $\Re\mathcal{I}$ denote the lexicographic product $(\Re^+, \mathit{Integer})$ where $\Re^+$ is the (lifted) domain of non-negative reals. Then we define

$$\mathit{precision}_L :: \mathcal{CC}^X_L \to \Re\mathcal{I}$$
$$\mathit{precision}_L\big(\, x \sqsubseteq \langle \overline{a_b}, c_d \rangle \,\big) = \big(\, \hat{a} \diamond_L c, b \diamond_B d \,\big)$$

where $\diamond_L :: \big\{\, (\hat{a}, c) \;\big|\; a, c \in L, a \preceq c \,\big\} \to \Re^+$ is a (system or user defined) strict monotonic

[2]The *lifted lattice* of L is $L \cup \{\bot_L, \top_L\}$ where, if the greatest lower bound of L does not exist, $\bot_L$ is a new element not in L such that $\forall a \in L, \bot_L \prec a$ and similarly, if the least upper bound of L does not exist, $\top_L$ is a new element not in L such that $\forall a \in L, a \prec \top_L$.

[3]Despite that $\overline{s}$ and t belong to different domains, s and t can be compared as, by applying the duality principle of lattices [5], both s and t belong to L^s.

function and $\diamond_B :: B \times B \to \{0,1,2\}$ is the strict monotonic function

$$\text{`]'} \diamond_B \text{`]'} \stackrel{\text{def}}{=} 2 \qquad \text{`]'} \diamond_B \text{`)'} \stackrel{\text{def}}{=} 1$$
$$\text{`)'} \diamond_B \text{`]'} \stackrel{\text{def}}{=} 1 \qquad \text{`)'} \diamond_B \text{`)'} \stackrel{\text{def}}{=} 0.$$

Observe that $precision_L$ is defined only on consistent constraints and thus the function $\diamond_L$ only needs to be defined when its first argument is less than or equal to the second. This function must be defined for each computation domain including any fictitious top or bottom elements.

Example 1 *Let Integer, $\Re$ and IntegerSet and let also $\Re^2 = \langle \Re, \Re \rangle$. Suppose that $i_1, i_2 \in Integer$, $r_1, r_2, w_1, w_2 \in \Re$ and $s_1, s_2 \in IntegerSet$ where $i_1 \preceq i_2$, $r_1 \preceq r_2$, $w_1 \preceq w_2$ and $s_1 \preceq s_2$. Then, by using infix notation,*

$$\hat{i_1} \diamond_{Integer} i_2 = i_2 - i_1,$$
$$\hat{r_1} \diamond_{\Re} r_2 = r_2 - r_1,$$
$$\widehat{\langle r_1, w_1 \rangle} \diamond_{\Re^2} \langle r_2, w_2 \rangle = +\sqrt{(r_2 - r_1)^2 + (w_2 - w_1)^2},$$
$$\hat{s_1} \diamond_{IntegerSet} s_2 = \#s_2 - \#s_1.$$

Assume that $i \in V_{Integer}$, $r \in V_{\Re}$, $y \in V_{\Re^2}$ and $s \in V_{IntegerSet}$. Then

$$precision_{Integer}(i \sqsubseteq \langle \overline{1_]}, 4_] \rangle) = (3.0, 2),$$
$$precision_{\Re}(r \sqsubseteq \langle \overline{3.5_)}, 5.7_) \rangle) = (2.2, 0),$$
$$precision_{\Re^2}(y \sqsubseteq \langle \overline{(2.0, 3.0)_]}, (3.4, 5.6)_] \rangle) = (2.95, 2),$$
$$precision_{IntegerSet}\ (s \sqsubseteq \langle \overline{\{\}_]}, \{3, 4, 5\}_) \rangle) = (3.0, 1).$$

We define the precision of the precision of a consistent simple stable constraint store $S \in \mathcal{SS}^X$ as the sum of the precisions of each of its elements (i.e., constraints) and where the sum in $\Re\mathcal{I}$ is defined as $(a_1, a_2) + (b_1, b_2) = (a_1 + b_1, a_2 + b_2)$.

3 Branching concepts

In the following we introduce some concepts that will be used in the rest of the paper. Also $L_<$ denotes any totally ordered lattice in $\mathcal{L}$, and if $\{c_1, \ldots, c_n\} \in \mathcal{SS}^X$ and $i \in \{1 \ldots, n\}$, then $\{c_1, \ldots, c_n\}[c_i/c']$ denotes $\{c_1, \ldots, c_{i-1}, c', c_{i+1}, \ldots, c_n\}$.

Definition 1 *(Divisibility) Let $c = x \sqsubseteq \overline{s}, t$ be a consistent interval constraint in $\mathcal{C}_L^X$. Then, c is* divisible *if $s \neq_{L^s} t$ and* non-divisible *otherwise. Let $S \in \mathcal{SS}^X$ be a consistent constraint store. Then S is* divisible *if there exists $c \in S$ such that c is divisible and* non-divisible *otherwise.*

Basically a non-divisible constraint has the form $x \sqsubseteq$ **[a,a]** that, as already said is a shorthand for $x =$ a where $x \in V_L$ and a $\in L$ for some $L \in \mathcal{L}$ (i.e., a non-divisible constraint may be viewed as an assignment of a constrained variable in a domain to a value belonging to that domain).

Proposition 1 *Let $X \in \wp_f(V_{\mathcal{L}})$.*

(1) Let also $c, c' \in \mathcal{C}_L^X$ such that $c \prec_{\mathcal{C}_L^X} c'$. Then, if c is consistent, c' is divisible.

(2) Let also $S, S' \in \mathcal{SS}^X$ such that $S \prec_s S'$. Then, if S is consistent, S' is divisible.

In [6][Definition 13] we defined the concept of *solution* for a constraint store as a consistent stable store that produces no more constraint narrowing by constraint propagation. Now we redefine this solution concept to capture the usual meaning of a solution as an assignment of values to variables that satisfies all the constraints. So as to distinguish the previous concept defined in [6][Definition 13] from the concept defined in this paper, we use the term *solution* to refer the concept already defined and the term *authentic solution* to refer the new concept defined in this paper.

Definition 2 *(Authentic solution) Let $C \in \wp_f(\mathcal{C}^X)$ be a constraint store for X and $R \in \mathcal{SS}^X$. Then, R is an* authentic solution for C *if R is both non-divisible and a solution for C, and $R' \in \mathcal{SS}^X$ is a* partial solution for C *if there exists an authentic solution R'' for C such that $R'' \prec_s R'$. In this case we say that R'* covers *R''.*

The set of all authentic solutions for C is denoted as $Sol_a(C)$.

Definition 3 *(Constraint store stack) Let* $P = (S_1, \ldots, S_\ell)$ *be any (possibly empty) sequence where* $S_i \in \mathcal{SS}^X$ *for* $1 \leq i \leq \ell$ *and* $\ell \geq 0$. *Then* P *is a* constraint store stack for X *if the operation push/2 over* P *is defined for any* $S \in \mathcal{SS}^X$ *as follows*

$$\begin{aligned}
&\mathbf{P}recondition : \{\ P = (S_1, \ldots, S_\ell)\ \} \\
&\quad push(P, S) \\
&\mathbf{P}ostcondition : \{\ P = (S_1, \ldots, S_\ell, S_{\ell+1}), \\
&\qquad\qquad S_{\ell+1} = S \text{ and } P \in Stack(X)\ \}.
\end{aligned}$$

where $Stack(X)$ *is the set of all constraint store stacks for* X, *and the operation top/1 over* P *is defined as:*

$$\begin{aligned}
&\mathbf{P}recondition : \{P = (S_1, \ldots, S_\ell) \text{ and } \ell > 0\} \\
&\quad top(P) = S \\
&\mathbf{P}ostcondition : \{S = S_\ell\}.
\end{aligned}$$

Let $P' = (S'_1, \ldots, S'_{\ell'})$ *be another constraint store stack for* X. *Then* $P \preceq_p P'$ *if and only if for all* $S_i \in P$ $(1 \leq i \leq \ell)$, *there exists* $S'_j \in P'$ $(1 \leq j \leq \ell')$ *such that* $S_i \preceq_s S'_j$. *In this case we say that* P' covers P.

4 The Branching Process

Branching [1] often involves two steps of choice usually called *variable ordering* and *value ordering*. The first step selects a constrained variable and the second one splits the domain associated to the selected variable in order to introduce a choice point. In this section we explain these choice steps by describing the main functions that define them.

4.1 Involved Functions

The *selecting function* provides a schematic heuristic for variable ordering.

Definition 4 *(Selecting function) Let* $S = \{c_1, \ldots, c_n\} \in \mathcal{SS}^X$. *Then*

$$choose :: \{S \in \mathcal{SS}^X \mid S \text{ is divisible}\} \to \mathcal{C}^X$$

is called a selecting function for X *if* $choose(S) = c_j$ *where* $1 \leq j \leq n$ *and* c_j *is divisible.*

When branching, some interval constraints need to be partitioned, into two or more parts, so as to introduce a choice point. We define a *splitting function* which provides a heuristic for value ordering.

Definition 5 *(Splitting function) Let* $L \in \mathcal{L}$ *and* $k > 1$. *Then*

$$split_L :: \mathcal{C}_L^X \to \underbrace{\mathcal{C}_L^X \times \ldots \times \mathcal{C}_L^X}_{k \text{ times}}$$

is called a k-ary splitting function for L *if, for all* $c \in \mathcal{C}_L^X$, *with* c *divisible, this function is defined* $split_L(c) = (c_1, \ldots, c_k)$ *such that the following properties hold:*

$$\begin{aligned}
&Completeness : \forall c' \prec_{\mathcal{C}_L^X} c \text{ with } c' \text{ non-divisible}, \\
&\qquad\qquad \exists i \in \{1, \ldots, k\}\ .\ c' \preceq_{\mathcal{C}_L^X} c_i. \\
&Contractance :\ c_i \prec_{\mathcal{C}_L^X} c,\ \forall i \in \{1, \ldots, k\}.
\end{aligned}$$

Lemma 1 *Let choose/1 be a selecting function for* X, $C \in \wp_f(\mathcal{C}^X)$, $S = (c_1, \ldots, c_n) \in \mathcal{SS}^X$ *a divisible constraint store,* $c_j = choose(S)$, $c_j \in \mathcal{C}_L^X$ *for some* $L \in \mathcal{L}$, $split_L/1$ *a k-ary splitting function for* L *and* $(c_{j1}, \ldots, c_{jk}) = split_L(c_j)$. *Then*

(a) $\forall i \in \{1, \ldots, k\} : S[c_j/c_{ji}] \prec_s S$;

(b) if $S' \in Sol_a(C)$ *and* $S' \prec_s S$, *then*

$$\exists i \in \{1, \ldots, k\} : S' \preceq_s S[c_j/c_{ji}].$$

4.2 Precision Map: a Normalization Rule

The precision map described in Section 2 also provides a way to normalize the selecting functions (i.e., the variable ordering) when the constraint system supports multiple domains.

Example 2 *The well known* first fail principle *chooses the variable constrained with the smallest domain. However, in systems supporting multiple domains it is not always clear which is the smallest domain (particularly if there are several infinite domains). In our framework, one way to "measure" the size of the domains is to use the precision map defined on each computation domain.*

For instance, suppose that $X = \{x_1, \ldots, x_n\}$ is a set of variables constrained, respectively, in $L_1, \ldots, L_n \in \mathcal{L}$ and that $S = \{c_1, \ldots, c_n\} \in \mathcal{SS}^X$ is any divisible constraint store for X where for each $i \in \{1, \ldots, n\}$, c_i is the simple interval constraint in S with constrained variable x_i. Here the first fail principle can be emulated by defining choose/1 to select the interval constraint with the smallest precision[4]*. We denote this procedure by $choose_{ff}$.*

P*recondition* :

$$\{S = \{c_1, \ldots, c_n\} \in \mathcal{SS}^X \text{ is divisible}\}$$
$$choose_{ff}(S) = c_j$$

P*ostcondition* : $\{j \in \{1, \ldots, n\}, c_j$ *is divisible and* $\forall i \in \{1, \ldots, n\} \backslash \{j\} : c_i$ *divisible* $\Longrightarrow$ $precision_{L_j}(c_j) \leq_{\Re\mathcal{I}} precision_{L_i}(c_i)\}$.

4.3 The Branching Operational Schema

In [6][Section 5], for some $C \in \wp_f(\mathcal{C}^X)$ and $S \in \mathcal{SS}^X$, we defined $solve(C, S)$, a generic *operational schema* for computing a solution (if it exists) for $C \cup S$ (this solution is returned in S). We then proved the correctness of this schema. To guarantee termination, we extended the schema, and called this extended schema $solve_\varepsilon/2$, with a parameter $\varepsilon \in \Re^+$ (i.e., a non-negative real number) that guaranteed termination. Now Figure 1 shows a generic schema that secures completeness of the interval constraint solving. This schema is complementary to that described in [6].

The schema requires the following parameters: a finite set $C \in \wp_f(\mathcal{C}^X)$ of interval constraints to be solved, a constraint store $S \in \mathcal{SS}^X$, a bound $p \in \Re\mathcal{I}$ and a non-negative real bound α. In addition to those properties of $solve_\varepsilon/2$ already declared and proved in [6], we state here some new properties:

Lemma 2 *Let $C \in \wp_f(\mathcal{C}^X)$, $S, S^f \in \mathcal{SS}^X$ and $\varepsilon \in \Re^+$. Suppose that S^f is the value of the constraint store S after a terminating execution of $solve_\varepsilon(C, S)$. Then,*

(a) $S^f \preceq_s S$;

(b) $\forall R \in Sol_a(C \cup S) : R \preceq_s S^f$;

(c) If $Sol_a(C \cup S)$ is not empty and S^f is non-divisible then $S^f \in Sol_a(C \cup S)$;

(d) If $\varepsilon = 0.0$ and S^f is non-divisible then $S^f \in Sol_a(C \cup S)$.

Property (a) ensures that the propagation procedure never gains values, property (b) guarantees that no solution covered by a constraint store is lost in the propagation process and properties (c) and (d) guarantee the computed answers are correct. There are a number of values and subsidiary procedures that are assumed to be defined externally to the main branch procedure shown in Figure 1:

- a selecting function $choose/1$ for X;
- a k-ary splitting function $split_L$ for each domain $L \in \mathcal{L}$ (for some integer $k > 1$);
- a precision map for each $L \in \mathcal{L}$;
- a constraint store stack P for X.

It is assumed that the external procedures have an implementation that terminates for all possible values.

Theorem 1 *(Properties of the $branch_\alpha/3$ schema) Let $C \in \wp_f(\mathcal{C}^X)$, $S \in \mathcal{SS}^X$, $\varepsilon, \alpha \in \Re^+$ and $p = \top_{\Re\mathcal{I}}$. Then, the following properties are guaranteed:*

1. Termination*: if $\alpha > 0.0$ and the procedure $solve_\varepsilon/2$ terminates for all values*[5] *then $branch_\alpha(C, S, p)$ terminates;*
2. Completeness: *if $\alpha = 0.0$ and the execution of $branch_\alpha(C, S, p)$ terminates, then the final state for the stack P contains all the authentic solutions for $C \cup S$;*
3. Approximate completeness: *if the execution of $branch_\alpha(C, S, p)$ terminates and $R \in Sol_a(C \cup S)$, then the final state for the stack P contains either R or a partial solution R' that covers R.*

[4] It is straightforward to include more conditions e.g., if c_i, c_k, c_j have the same (minimum) precision, the "left-most" domain can be chosen i.e., $c_{minimum(i,k,j)}$.

[5] Observe that termination of this procedure is always guaranteed if $\varepsilon > 0.0$ -see Theorem 2 in [6].

procedure $branch_\alpha(C, S, p)$
begin
$solve_\varepsilon(C, S)$; (1)
if S is consistent **then** (2)
  if (S is non-divisible **or** (3)
  $p < \top_{\Re\mathcal{I}}$ **and** $p - precision(S) \leq (\alpha, 0)$) **then**
    $push(P, S)$; (4)
  else (5)
    $c_j \leftarrow choose(S)$; (6)
    $(c_{j1}, \ldots, c_{jk}) \leftarrow split_{L_j}(c_j)$, (7)
      %% where $c_j \in \mathcal{C}^X_{L_j}$ and $L_j \in \mathcal{L}$;
    $\left.\begin{array}{ll} branch_\alpha(C, S[c_j/c_{j1}], precision(S)) & \vee \\ \ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots & \vee \\ branch_\alpha(C, S[c_j/c_{jk}], precision(S)); & \end{array}\right\}$ (8)
  endif
endif
endprocedure

Figure 1: Schema for interval constraint solving

4. Correctness: *if* $\alpha = 0.0$ *and* $\varepsilon = 0.0$, *the stack* P *is initially empty and the execution of* $branch_\alpha(C, S, p)$ *terminates with* R *in the final state of* P, *then* $R \in Sol_a(C \cup S)$.

5. Approximate correctness or control on the result precision: *If* P_{α_1} *and* P_{α_2} *are non-empty constraint store stacks for* X *resulting from any terminating execution of* $branch_\alpha(C, S, p)$ *(where initially* P *is empty) when* α *has the values* α_1 *and* α_2, *respectively, and* $\alpha_1 < \alpha_2$ *then*

$$P_{\alpha_1} \preceq_p P_{\alpha_2}.$$

(In other words, the set of (possibly partial) solutions in the final state of the stack is dependent on the value of α *in the sense that lower* α, *better the set of solutions.)*

Observe that, just as for the bound ε in the $solve_\varepsilon/2$ procedure, the bound α also guarantees termination and allows the precision of the results to be controlled. Note also that this schema can be used for any set of computation domains for which a splitting function and precision map are defined.

5 Examples

To illustrate the $branch_\alpha/3$ schema presented in the preceding section, several instances of it are given for some well-known domains of computation. In the following, $branch_L$ denotes an instance of the schema $branch_\alpha/3$ for solving interval CSPs defined on $L \subseteq \mathcal{L}$.

For simplicity, in all the examples we assume as selecting function the function $choose_{ff}$ as defined in Example 2. For each instance $branch_L$, we assume the definitions for $\diamond_L/2$ shown in Example 1 so that the precision map for L is then implicitly defined, and indicate possible definitions for the splitting function.

(1) $branch_{Integer}$: the finite domain.

$$split_{Integer}(x \sqsubseteq \{_1a_1, a_k\}_2) = (\ x \sqsubseteq \{_1a_1, a_1], x \sqsubseteq [a_1 + 1, a_k\}_2\ ).$$

Observe that $split_{Integer}$ is defined as a naive enumeration strategy in which values are chosen from left to right.

(2) $branch_\Re$: a continuous domain.

$$split_\Re(x \sqsubseteq \{_1a, b\}_2) = (x \sqsubseteq \{_1a, c], x \sqsubseteq (c, b\}_2)$$

where $a \preceq_\Re c \prec_\Re b$ e.g., if $c = \frac{b-a}{2.0}$ we have a usual real interval division at the mid point.

(2) $branch_{IntegerSet}$: finite Sets of integers.

$$split_{Integer\ Set}\ (x \sqsubseteq [a, b]) = (x \sqsubseteq [a, b \backslash \{c\}], x \sqsubseteq [a \cup \{c\}, b]),\ \text{where } c \in b \backslash a.$$

The schema also supports cooperative instances that solve CSP's defined on multiple domains. This is done by mixing together several instances of the schema.

6 Solving Optimization Problems

The schema in Figure 1 can be adapted for alternative objectives, e.g., solving constraint optimization problem (COPs). This can be done by means of three new subsidiary functions.

Definition 6 *(Subsidiary functions and values) Let $L_< \in \mathcal{L}$ be a totally ordered domain*[6]*. Then we define*

- *a* cost function*, $fcost :: \mathcal{SS}^X \to L_<$;*
- *an* ordering relation*,*

$$\diamond :: L_< \times L_< \in \{>, <, =\};$$

- *a* cost bound*, $\delta \in L_<$.*

The *extended branching schema*, $branch_{\alpha+}/3$, is obtained by replacing Line 4 in Figure 1 (i.e., $push(P, S)$) with:

if $fcost(S) \diamond \delta$ **then** (4*)
  $\delta \leftarrow fcost(S)$;
  $push(P, S)$;
endif

Theorem 2 *(Properties of the $branch_{\alpha+}/3$ schema) Let $C \in \wp_f(\mathcal{C}^X)$, $S \in \mathcal{SS}^X$, $\varepsilon, \alpha \in \Re^+$ and $p = \top_{\Re\mathcal{I}}$. Suppose that the procedure $solve_\varepsilon/2$ terminates for all values*[7]*. Then, the following properties are guaranteed:*

1. Termination*: if $\alpha > 0.0$ then the execution of $branch_{\alpha+}(C, S, p)$ terminates;*
2. *If fcost is a constant function with value δ and $\diamond$ is $=$, then all properties shown in Theorem 1 hold for the execution of $branch_{\alpha+}(C, S, p)$.*
3. Soundness on optimization*: If at least one authentic solution with a cost higher than $\bot_{L_<}$ (resp. lower than $\top_{L_<}$) exists for $C \cup S$, $\alpha = 0.0$, $\diamond$ is $>$ (resp. $<$), $\delta = \bot_{L_<}$ (resp. $\top_{L_<}$), the stack P is initially empty and the execution of $branch_{\alpha+}(C, S, p)$ terminates with P non-empty, then the element on the top of P is the first authentic solution found that maximizes (resp. minimizes) the cost function.*

Unfortunately, if $\alpha > 0.0$, we cannot guarantee that the top of the stack contains an authentic solution or even a partial solution for the optimization problem. However, if the cost function $fcost/1$ is monotonic, solutions can be compared.

Theorem 3 *(Approximate soundness) Suppose that, for $i \in \{1, 2\}$, P_{α_i} is the constraint store stack resulting from the execution of $branch_{\alpha_i+}(C, S, p)$ where $\alpha_i \in \Re^+ \cup \{0.0\}$. Then, if $\alpha_1 < \alpha_2$ the following property hold.*

If P_{α_1} and P_{α_2} are not empty, and $top(P_{\alpha_2})$ is an authentic solution or covers a solution for $C \cup S$, then, if $fcost/1$ is monotonic and $\diamond$ is $<$ (i.e., a minimization problem),

$$fcost(top(P_{\alpha_1})) \preceq_{L_<} fcost(top(P_{\alpha_2})),$$

and, if $fcost/1$ is anti-monotone and $\diamond$ is $>$ (i.e.,a maximization problem),

$$fcost(top(P_{\alpha_1})) \succeq_{L_<} fcost(top(P_{\alpha_2})).$$

A direct consequence of this theorem is that by using a(n) (anti-)monotone cost function, lower α is, better the (probable) solution is. Moreover, decreasing α is a means to discard approximate solutions. For instance, in a minimization problem, if

$$fcost(top(P_{\alpha_1})) \succ_{L_<} fcost(top(P_{\alpha_2}))$$

with $fcost/1$ monotonic, then, by the *approximate soundness* property it is deduced that $top(P_{\alpha_2})$ cannot be an authentic solution or cover an authentic solution.

6.1 Different Ways to Solve the Instances

In this section, we explain how the choice of the instantiation of the additional global functions and parameters in the definition of $branch_{\alpha+}/3$ determines the method of solving for a set of interval constraints i.e., the schema $branch_{\alpha+}/3$ allows a set of interval constraints to be solved in many different ways, depending on the values for *fcost*, δ and $\diamond$.

Theorem 2(2) has shown that to solve classical CSPs, *fcost* should be defined as the constant function[8] δ and the parameter $\diamond$ should have the value $=$. Moreover, Theorem 2(3) has shown that a CSP is solved as a COP by

[6] Normally $L_<$ would be $\Re$.

[7] Again note that termination of this procedure is always guaranteed if $\varepsilon > 0.0$ -see Theorem 2 in [6].

[8] Usually $\delta \in \Re$.

instantiating $\diamond$ as either $>$ (for maximization problems) or $<$ (for minimization problems). In all cases, the value δ should be instantiated to the initial cost value from which an optimal solution must be found. Some possible instantiations are summarized in Table 1 where Column 1 indicates the type of CSP, Column 2 gives any conditions on the cost function, Column 3 gives the range of the cost function (usually, this is $\Re$), Columns 4 gives the initial definition of the $\diamond$ operator, and Columns 5 gives the initial value for δ.

CSP Type	*fcost*	$L_<$	$\diamond$	δ
Classical CSP	constant	$\Re$	$=$	$fcost(S)$
Minimization COP	any	$\Re$	$<$	$\top_{\Re}$
Maximization COP	any	$\Re$	$>$	$\bot_{\Re}$
Max-Min COP	any	$\Re \times \Re$	$<$	$\top_{\Re\times\Re}$

Table 1: CSP solving dependency on parameter instantiation

In contrast to typical COPs that usually maintain a fixed criteria (i.e., either maximization or minimization of the cost function) and a single lower or upper bound, our schema also permits a mix of the maximization and minimization criteria (or even to give priority to some criteria over others). This is the case (see Row 4 of Table 1) when $L_<$ is a compound domain and the ordering in $L_<$ determines how the COP will be solved.

Example 3 *Let $C \in \wp_f(\mathcal{C}^X)$ be a set of interval constraints to be solved as a COP, $L_<$ the domain $\Re^2 = \Re \times \Re$ with ordering*

$$(a,b) < (c,d) \iff (a < c \wedge b \geq d) \vee (a \leq c \wedge b > d),$$

and $fcost :: \mathcal{SS}^X \to L_<$ a cost function on $\Re^2$ defined for any $S \in \mathcal{SS}^X$ as

$$fcost(S) = (fcost_1(S), fcost_2(S))$$

where $fcost_1, fcost_2 :: \mathcal{SS}^X \to \Re$ are cost functions defined on $\Re$. Then, if δ and $\diamond$ are initialized respectively to $<$ and $\top_{\Re^2}$ (as shown in Row 4 of Table 1), C is solved by minimizing $fcost_1$ and maximizing $fcost_2$.

On the other hand, if $<$ is defined lexicographically on $\Re^2$, i.e.,

$$(a,b) < (c,d) \iff a < c \vee a = c \wedge b < d,$$

C is solved by giving priority to the minimization of $fcost_1$ over the minimization of $fcost_2$.

For example, suppose $Sol_a(C) = \{S_1, S_2, S_3\}$ and $fcost(S_1) = (1.0, 5.0)$, $fcost(S_2) = (3.0, 1.0)$ and $fcost(S_3) = (1.0, 8.0)$. Suppose also that these solutions have been found by a terminating execution of the $branch_{\alpha+}/3$ schema where $\diamond \equiv <$ and initially $\delta \equiv \top_{\Re^2}$ and that the sequence in which the solutions are found in the search tree is (S_1, S_2, S_3).

Consider the first ordering defined above for $\Re^2$. When S_1 is found, line 4 of the schema is executed with $\delta = (\top_{\Re}, \bot_{\Re})$ (i.e., with $\delta = \top_{\Re^2}$ as shown in Row 4 of Table 1) and as consequence S_1 is pushed on the stack P. Afterwards, S_2 is found and line 4* is executed with $\delta = fcost(S_1) = (1.0, 5.0)$. As $fcost(S_2) \not< (1.0, 5.0)$, S_2 is not pushed on the stack. Next S_3 is found and again line 4* is executed with $\delta = fcost(S_1) = (1.0, 5.0)$. As $fcost(S_3) < (1.0, 5.0)$ then S_3 is pushed on the stack so that the top of the new stack contains S_3. Note S_3 minimizes the first component of the cost and maximizes the second component.*

Consider next the lexicographic ordering for the domain $\Re^2$. When S_1 is found, line 4 is executed with $\delta = (\top_{\Re}, \top_{\Re})$ (i.e., with $\delta = \top_{\Re^2}$ as shown in Row 4 of Table 1) and as consequence S_1 is pushed on the stack P. Afterwards, S_2 is found and line 4* is executed with $\delta = fcost(S_1) = (1.0, 5.0)$. As $fcost(S_2) \not< (1.0, 5.0)$ then S_2 is not pushed on the stack. Finally S_3 is found and again line 4* is executed with $\delta = fcost(S_1) = (1.0, 5.0)$. As $fcost(S_3) \not< (1.0, 5.0)$, S_3 is not pushed on the stack and the top of the stack contains S_1. In this case, S_1 minimizes the first component and only if the values of the first components are equal, minimizes the second component.*

7 Related Work

Constraint solving algorithms have received intense study from many researchers, although the focus was on developing new and more efficient methods to solve classical CSPs [7, 23] and partial CSPs [8, 14]. See [13, 19, 20, 22] for more information on constraint solving algorithms and [12, 17] for selected comparisons.

Most of the work existing in the literature about the branching step is focused on the discrete domain and, in this case, branching is usually called *labeling* [21]. Labeling consists of assigning values (i.e., the instantiation) to the constrained variables and, by a backtracking search, to find a solution (if it exists) for the CSP. The order in which variables and values are instantiated will have a significant influence on the shape of the search tree and thus the performance of the solution [1].

On infinite domains, labeling is rarely applied as for FD. Of course there are exceptions such as that shown in [16, 15] that applied labeling to process the solutions on infinite and continuous domains. Before applying labeling, the only values a variable can take are roots of an univariate polynomial so that in fact only discrete and finite domains are considered.

Traditionally, on the continuous domain (i.e., the real domain) the branching process consists of splitting (usually in two parts) the domain of some variable(s) so as to continue with the search for a solution in each of the derived partitions. This is the process followed in well known systems such as CLP(BNR) [18] and CLIP [11]. These systems provide interval constraint solving on which a real variable has associated an interval (in the usual meaning of set theory) and a classical strategy of "divide and conquer" in the solving of problems involving real numbers is usually employed. When no more propagation is possible, the interval solver uses a sort of domain splitting to return each answer. This method is called *split-and-solve* [2]. The *split-and-solve* method repeatedly selects a variable, splits its associated interval into two or more parts and uses backtracking to look for solutions in each partition. Of course, there is the necessity of a termination test that avoids the infinite splitting of ranges (at least theoretically because in practice the real domain is finite since the precision of a machine is finite). Particularly, CLP(BNR) extends this strategy to the Boolean and integer domains.

8 Concluding Remarks

This paper it is an attempt to find general principles for the branching process in interval constraint solving. The branching schema provided here is a generic schema for solving sets of interval constraints on finite and continuous domains as well on multiple domains and it is useful to prove and devise generic properties of interval constraint solving.

Our branching schema generalizes the well known split-and-solve method of the CLP(BNR) system [2] to any domain with lattice structure what means that it is valid for both classical domains (i.e., real, integers, Boolean and sets) and new (possibly combined) domains. In this generalization, we propose an interval branching schema that extends the generic and cooperative interval propagation schema described in [6]. This extension provides a generic schema for interval constraint solving that allows problems defined on any set of lattices to be solved in terms of interval constraints.

To achieve this, we have first defined the concept of authentic solution as an assignment of values to variables that satisfies all the constraints. Then, by using a schematic formulation for the branching process, we have indicated which properties of the main procedures involved in branching are responsible for the key properties of interval constraint solving. Then we have extended the schema for optimization and have shown that, in some cases, the methods for solving CSPs depend on the ordering of the range of the cost functions.

We have also proved key properties such as correctness and completeness and shown how termination may be guaranteed by means of a *precision map* similar to that defined for the propagation schema described in [6]. Moreover, by means of an example, we have also shown how the precision map is a means to normalize the heuristic for variable ordering on systems supporting multiple domains (e.g., cooperative systems).

Our branching schema can be used for most existing constraint domains (finite or continuous) and, as for the propagation framework

described in [6], is also applicable to multiple domains and cooperative systems. Moreover, our branching schema can explain the behavior of a number of existing interval constraint systems such as such as clp(FD) [4], clp(B) and clp(B/FD) [3], DecLic [10], CLIP [11], Conjunto [9] or CLP(BNR) [2];

An extended version of the paper with proofs of propositions, lemmas and theorems can be found in *http://www.lcc.uma.es/∼afdez/Papers/*.

References

[1] K.R. Apt. *Principles of constraint programming*. Cambridge University Press, 2003.

[2] F. Benhamou and W.J. Older. Applying interval arithmetic to real, integer and Boolean constraints. *The Journal of Logic Programming*, 32(1):1–24, July 1997.

[3] P. Codognet and D. Diaz. clp(B): combining simplicity and efficiency in Boolean constraint solving. In *PLILP'94*, number 844 in LNCS, pages 244–260, Madrid, Spain, 1994. Springer-Verlag.

[4] P. Codognet and D. Diaz. Compiling constraints in clp(FD). *The Journal of Logic Programming*, 27(3):185–226, 1996.

[5] B.A. Davey and H.A. Priestley. *Introduction to lattices and order*. Cambridge University Press, Cambridge, England, 1990.

[6] A. J. Fernández and P.M. Hill. An interval constraint system for lattice domains. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 26(1):1–46, 2004.

[7] E.C. Freuder and P. Hubbe. Extracting constraint satisfaction subproblems. In *IJCAI'95*, pages 548–557, Québec, Canada, August 1995. Morgan Kaufman.

[8] E.C. Freuder and R.J. Wallace. Partial constraint satisfaction. *Artificial Intelligence*, 58(21-70):21–70, 1992.

[9] C. Gervet. Interval propagation to reason about sets: definition and implementation of a practical language. *Constraints*, 1(3):191–244, 1997.

[10] F. Goualard, F. Benhamou, and L. Granvilliers. An extension of the WAM for hybrid interval solvers. *The Journal of Functional and Logic Programming*, 1999(1):1–36, April 1999.

[11] T.J. Hickey. CLIP: a CLP(Intervals) dialect for metalevel constraint solving. In E. Pontelli and V.Santos Costa, editors, *PADL'2000*, number 1753 in LNCS, pages 200–214, Boston, USA, 2000. Springer-Verlag.

[12] G. Kondrak and P. Van Beek. A theoretical evaluation of selected backtracking algorithms. *Artificial Intelligence*, 89(1-2):365–387, January 1997.

[13] V. Kumar. Algorithms for constraint satisfaction problems: a survey. *AI Magazine*, 13(1):32–44, Spring 1992.

[14] P. Meseguer and J. Larrosa. Constraint satisfaction as global optimization. In *IJCAI'95*, pages 579–585, Québec, Canada, August 1995. Morgan Kaufman.

[15] E. Monfroy. *Solver collaboration for constraint logic programming*. PhD thesis, Centre de Recherche en Informatique de Nancy, INRIA-Lorraine, November 1996.

[16] E. Monfroy, M. Rusinowitch, and R. Schott. Implementing non-linear constraints with cooperative solvers. Research Report 2747, Centre de Recherche en Informatique de Nancy, INRIA-Lorraine, December 1995.

[17] B.A. Nadel. Constraint satisfaction algorithms. *Computational Intelligence*, 5:188–224, 1989.

[18] W. Older and F. Benhamou. Programming in CLP(BNR). 1st International Workshop on Principles and Practice of Constraint Programming (PPCP'93), Informal Proceedings, pages: 228-238, Brown University, Newport, Rode Island, 1993.

[19] Z. Ruttkay. Constraint satisfaction-a survey. *CWI Quaterly*, 11(2-3):163–214, 1998.

[20] B.M. Smith. A tutorial on constraint programming. Research Report 95.14, University of Leeds, School of Computer Studies, England, April 1995.

[21] P. Van Hentenryck. *Constraint satisfaction in logic programming*. The MIT Press, Cambridge, MA, 1989.

[22] P. Van Hentenryck. Constraint solving for combinatorial search problems: a tutorial. In U. Montanari and F. Rossi, editors, *CP'95*, number 976 in LNCS, pages 564–587, Cassis, France, 1995. Springer-Verlag.

[23] R.J. Wallace. Why AC-3 is almost always better than AC-4 for establishing arc consistency in CSPs. In R. Bajcsy, editor, *(IJCAI'93)*, pages 239–247, Chambéry, France, 1993. Morgan Kaufmann.