Constraints: Modeling & Propagation

Lecture 07, 2015-04-20

Christian Schulte
cschulte@kth.se

Software and Computer Systems
School of Information and Communication Technology
KTH – Royal Institute of Technology
Stockholm, Sweden
How To Design a Propagator?

- What do we know?
- What should a propagator prune?
  - What is it allowed to prune?
  - What is it not allowed to prune?
- How do we implement a propagator?
- What else should we think of?

Example: \( x \leq y \)
Propagator for ≤

- Propagator \( p_{≤} \) for \( x ≤ y \)
  \[
p_{≤} (s) = \begin{cases} 
    x \rightarrow \{ \ n \in s(x) \mid n ≤ \text{max}(s(y)) \}, \\
    y \rightarrow \{ \ n \in s(y) \mid n ≥ \text{min}(s(x)) \}\end{cases}
  \]

- good one: \( \text{es}(p_{≤}) = \{ \text{max}(y), \text{min}(x) \} \)
- but also: \( \text{es}(p_{≤}) = \{ \text{any}(y), \text{any}(x) \} \)
Overview
Common Constraints

- Common constraints
  - reified, arithmetic, element, distinct, regular, …
  - …and other global constraints

- How are they used for modeling
- How are they propagated
- What makes them global
- Propagation strength
Linear Equality

- Propagator for

\[ \sum_{i=1}^{n} a_i x_i = d \]

where \( a_i, d \) integers, \( a_i \neq 0 \)

- How to propagate cheaply bounds information?
  - for each variable \( x_i \) consider how small and how large it possibly can be
  - restrict us here to \( ax + by = d \)
Element Constraint

- **Element constraint** $a[[x]] = y$
  - array of integers $a$
  - variables $x$ and $y$
  - value of $y$ is value of $a$ at $x$-th position
  - in particular: $0 \leq x < \text{elements in } a$

- **In Gecode**
  - `element(*this, a, x, y);`
  - also for arrays of variables
Distinct: Basic Question

- Propagating in domain-consistent fashion
  - need to consider all solutions
  - naïve: too much memory, too much time

- Is there a better way
  - in general no
  - but for particular constraints: yes!
  - but element is simple, for more difficult?
Golomb Rulers

- Find $n$ ticks $t_i$ on ruler such that
  - distance between ticks pairwise distinct
  - length of ruler minimal

- Extremely hard problem
  - applications in crystallography, …
Cardinality Constraint

- Distinct: variables can take a particular value at most once
- Generalization: lower and upper bounds on how often a value can be taken by variables
- Use: shift assignment
  - worker: variables describe shift to work
  - boss: sets minimum and maximum requirements on workers per shift
Channeling/Assignment Constraints

- Suppose variables $x_i$ and $y_i$ ($0 \leq i < n$)
- channel($x_i$, $y_i$) holds, iff
  \[ x_i = j \iff y_j = i \quad (0 \leq i < n) \]

- Use: dual models (permutation problems)
  - queen $i$ is in row $j$ \iff row $j$ contains queen $i$
  - post constraints on both set of variables
Lex Constraints

- Suppose variables $x_i$ and $y_i$ (0 $\leq$ $i$ < $n$)
- $\text{lex}(x_i, y_i)$ holds, iff
  $$(x_0, \ldots, x_{n-1})$$ lexicographically smaller than $$(y_0, \ldots, y_{n-1})$$

- Use: symmetry breaking
  - in particular: matrix models (magic square)
Regular Constraint

- Suppose variables $x_i$ ($0 \leq i < n$)
- regular($x_i$, $r$) holds, iff
  - the string $x_0 \ldots x_{n-1}$ forms a word from the language of the regular expression $r$

- Use: describe patterns formed by variable sequences
  - assignment!
Goal

- Understand and apply
  - modeling abstractions
  - modeling techniques

- Understand propagation techniques
Reification
Reification

- Reified propagator for a constraint $c$ and control variable $b$

\[ c \iff b=1 \land b \in \{0,1\} \]

- if $c$ holds \quad \Rightarrow \text{propagate } b=1
- if $c$ does not hold \quad \Rightarrow \text{propagate } b=0
- if $b=1$ holds \quad \Rightarrow \text{propagate } c
- if $b=0$ holds \quad \Rightarrow \text{propagate } \neg c
Reified Propagators

- A propagator $p$ is a reified propagator for the constraint $c$ and the control variable $b$, iff

  \[ a \in c \iff p(\text{store}(a))(b) = \{1\} \]
  \[ a \notin c \iff p(\text{store}(a))(b) = \{0\} \]

  for all assignments $a$
Example: Reification

- In Gecode, reified versions typically take additional Boolean variable

- Example: reification of $x < y$

  ```
  BoolVar b(*this,0,1);
  rel(this, x, IRT_LE, y, b);
  ```
Combination with Reification

- Arbitrary Boolean connectives
  - disjunction, conjunction, negation
  - implication, equivalence

- Counting

- Requires all or nothing approach
  - reification of $c_1 \land c_2 \lor c_3$
  - all of $c_1$, $c_2$, $c_3$ must be reified
  - sacrifices propagation!
Reification is Problematic

- Consider for some \( c \)
  \[ x = y + 1 \land y = x + 1 \lor c \]
  reification:
  \[ b_1 = (x = y + 1) \quad b_2 = (y = x + 1) \]
  \( b, b_1, b_2 \in \{0,1\} \)
  \[ b = b_1 \times b_2 \]

- Problem: \( b \) not assigned 0 by propagation!
  - each propagator propagates individually
  - no propagation through shared variables
  - reification is not propagation preserving
Compositional Approaches

- Few approaches known
Example for Reification

- \( z \) is the number of elements in array \( x \) (of size \( n \)) that are equal to \( y \)

```c
BoolVarArgs e(n);
for (int i=0; i<n; i++) {
    BoolVar b(*this,0,1);
    rel(this, x[i], IRT_EQ, y, b);
    e[i]=b;
}
linear(*this, e, IRT_EQ, z);
```
Example Problem: Photo

- Five persons
  alice, bob, carl, deb, evan
  take group photo with preferences
  alice \Leftrightarrow carl \quad bob \Leftrightarrow evan
  carl \Leftrightarrow deb \quad carl \Leftrightarrow evan
  deb \Leftrightarrow alice \quad deb \Leftrightarrow evan
  evan \Leftrightarrow alice \quad evan \Leftrightarrow bob

- Find placement with maximal satisfaction of preferences!
Photo: Variables

- Variables:
  - for each person: position (0 to number of persons - 1)
  - for each preference: 0/1 variable, 1 if preference satisfied
  - total satisfaction: sum of all preference variables (0 to number of preferences)
All positions must be distinct

Preference $A \Leftrightarrow B$ is satisfied, if position for $A$ is next to position for $B$
  - $A$ directly left to $B$, or
  - $B$ directly left to $A$

Total satisfaction sum of all preference variables
  - can be seen as soft constraints
Photo: Placement

- Assume a next to b is preference
  - position of a is $pa = pos[a]$  
  - position of b is $pb = pos[b]$  
- a next to b $\iff$  
  - a left of b or b left of a  
    - use Boolean exclusive or  
- a left to b $\iff$ $pa + 1 = pb$  
  - use reified linear equation
Photo: Improvements

- See examples in Gecode
  - case study in MPG
- Order?
- Better heuristic?
  - pick popular persons first (most constrained)
- Symmetry breaking?

Try yourself!
Summary: Reification

- Simple method for combining constraints
- Can sacrifice propagation
- Used to model
  - "soft" constraints or preferences
  - Boolean combination of constraints
Propagation Strength
Consistency: Intuition

- Statements after propagation
- Bounds consistency
  - for smallest and largest value, there must exist solution
- Domain consistency
  - each value must be contained in solutions
Domain Consistency

A propagator $p$ with $\text{var}(p) = (x_1, \ldots, x_k)$ is **domain-consistent** for constraint $c$, iff for all non-failed stores $s$

- if $n_i \in p(s)(x_i)$ then
- exist $n_j \in p(s)(x_j)$ ($i \neq j$) with
  \[
  (n_1, \ldots, n_k) \in \text{sol}(c)
  \]
Domain Consistency

- Restrict to values in solutions
- Example: \( x = 3y + 5z \)
  \[ s(x) = \{2..7\} \quad s(y) = \{0..2\} \quad s(z) = \{-1..2\} \]
- Solutions
  \((x, y, z): (3, 1, 0), (5, 0, 1), (6, 2, 0)\)
- Resulting store
  \(s'(x)\)
Domain Consistency

- Restrict to values in solutions
- Example: $x = 3y + 5z$
  \[ s(x) = \{2..7\} \quad s(y) = \{0..2\} \quad s(z) = \{-1..2\} \]

- Solutions
  \[(x, y, z): \quad (3,1,0), (5,0,1), (6,2,0)\]

- Resulting store
  \[s'(x) = \{3,5,6\}\]
Domain Consistency

- Restrict to values in solutions
- Example: $x = 3y + 5z$
  
  $s(x) = \{2..7\}$  
  $s(y) = \{0..2\}$  
  $s(z) = \{-1..2\}$

- Solutions
  
  $(x, y, z): (3,1,0), (5,0,1), (6,2,0)$

- Resulting domain
  
  $s'(x) = \{3,5,6\}$  
  $s'(y) = \{0,1,2\}$  
  $s'(z) = \{0,1\}$
Assumptions

- In the following we assume that the universe is some finite subset of the integers, we will write for
  - finite subset $F$ (=Universe)
  - set of integers $Z$
  - set of reals $R$

- Consistency notions can be generalized to finite, total orders
Ranges

- **Function**
  \[ \text{range} \in 2^F \rightarrow 2^F \]
  \[ \text{range}(f) = \{\min(f), \ldots, \max(f)\} \]

- **For example**
  \[ \text{range}\{1,3,5\} = \{1,2,3,4,5\} \]

- **A subset** \( f \) **of** \( F \) **is called range, if**
  \[ \text{range}(f) = f \]
Bound Consistency

- A propagator \( p \) with \( \text{var}(p) = (x_1, \ldots, x_k) \) is \textit{bound-consistent} for constraint \( c \), iff for all non-failed stores \( s \)

  \[
  \text{if } n_i \in \{\min p(s)(x_i), \max p(s)(x_i)\} \text{ then exist } n_j \in \text{range}(p(s)(x_j)) (i \neq j) \text{ with } (n_1, \ldots, n_k) \in \text{sol}(c)
  \]
Simple Properties

- A domain-consistent propagator is idempotent
- A bounds-consistent propagator is not necessarily idempotent

Proof? Try it!
Linear Equality
Linear Equality

- Propagator for

\[ \sum_{i=1}^{n} a_i x_i = d \]

where \( a_i, d \) integers, \( a_i \neq 0 \)

- How to propagate cheaply bounds information?

  - for each variable \( x_i \) consider how small and how large it possibly can be
  - restrict us here to \( ax + by = d \)
Floor and Ceiling

\[ \lfloor x \rfloor \] (read: floor of \( x \)) is greatest integer \( k \) such that: \( k \leq x \)

- example: \( \lfloor 3.5 \rfloor = 3 \) \( \lfloor -3.5 \rfloor = -4 \)

\[ \lceil x \rceil \] (read: ceiling of \( x \)) is smallest integer \( k \) such that: \( k \geq x \)

- example: \( \lceil 3.5 \rceil = 4 \) \( \lceil -3.5 \rceil = -3 \)
Propagating Linear Equality

- Rewrite for $x$
  
  \[ ax + by = d \iff ax = d - by \iff x = (d - by)/a \]

- Propagate
  
  \[ x \leq \left\lceil \max\{(d - bn)/a \mid n \in s(y)\} \right\rceil \]
  
  and
  
  \[ x \geq \left\lfloor \min\{(d - bn)/a \mid n \in s(y)\} \right\rfloor \]
Propagating Linear Equality

- Computing
  \[
  m = \max\{(d - bn)/a \mid n \in s(y)\}
  \]
- If \(a > 0\) then
  \[
  m = \max\{(d - bn) \mid n \in s(y)\} / a
  \]
- If \(a < 0\) then
  \[
  m = \min\{(d - bn) \mid n \in s(y)\} / a
  \]
Propagating Linear Equality

- Computing \((a > 0)\)
  \[ m = \max\{(d - bn) \mid n \in s(y)\}/a \]
  \[ = (d - \min\{bn \mid n \in s(y)\})/a \]

- If \(b > 0\), then
  \[ m = (d - b \times \min s(y))/a \]

- If \(b < 0\), then
  \[ m = (d - b \times \max s(y))/a \]
General Setup

- Repeat until fixpoint
  - propagate for each variable $x_i$

- Speed up: compute once

\[
\begin{align*}
u &:= \max\left\{ d - \sum_{i=1}^{n} a_i n_i \mid n_i \in s(x_i) \right\} \\
l &:= \min\left\{ d - \sum_{i=1}^{n} a_i n_i \mid n_i \in s(x_i) \right\}
\end{align*}
\]

- Reuse by removing term for $x_i$
- Refer to propagator by $p_-$
Questions

- Is it necessary to perform several iterations?
  - yes, otherwise it is not idempotent
  - reason: non-unit coefficients

- What does $p_-$ compute?
  - is it bounds-consistent?
Example

Example: $x = 3y + 5z$

$s(x) = \{2..7\}$  $s(y) = \{0..2\}$  $s(z) = \{-1..2\}$

propagator:

$p_x(s)(x) = \{\min s(3y + 5z) \ldots \max s(3y + 5z)\}$

Resulting domain

$s'(x) = \{2..7\}$  $s'(y) = \{0..2\}$  $s'(z) = \{0..1\}$

- different from bounds propagation!
- should be 3 and 6!
What Is Computed?

- Algorithm just considers existence of real solutions
  - bounds-consistency defined for integer solutions only
- Possible: introduce new notion
  - $\mathbb{R}$-bounds consistency
  - Allow constraints to be defined by solutions over the reals
More Details

- Apt's book: Section 6.4
- Paper

Summary: Propagation Strength

- Propagators can have different propagation strengths
- Interesting classes
  - domain-consistent propagators
  - bounds-consistent propagators
- Typical propagators for linear arithmetic are *not* bounds-consistent
  - but close: not for integers, but for reals
Element Constraint

Constraints defined by extension
Modeling Price

- Suppose variable modeling location in warehouse
  - values model good to be stored at location
  - different goods have different prices
- How to propagate the price while the variable is not yet assigned a good?
- Very common: map variable to variable according to given values
Example

- Assume goods represented by numbers 0, 1, 2, 3

- Prices
  - good 0 price 10
  - good 1 price 15
  - good 2 price 5
  - good 3 price 12
Model by Reification

\[
\text{BoolVar } b_0(*\text{this}, 0, 1); \\
\ldots
\]

rel(*this, g, IRT_EQ, 0, b0);
rel(*this, p, IRT_EQ, 10, b0);
rel(*this, g, IRT_EQ, 1, b1);
rel(*this, p, IRT_EQ, 15, b1);
rel(*this, g, IRT_EQ, 2, b2);
rel(*this, p, IRT_EQ, 5, b2);
rel(*this, g, IRT_EQ, 3, b3);
rel(*this, p, IRT_EQ, 12, b3);
"b0 + b1 + b2 + b3 = 1";

- Tedious: several goods can have same price…
- Inefficient: too many propagators…
- Propagation: if propagators run to fixpoint, domain-consistency!
The Element Constraint

- Element constraint $a[[x]] = y$
  - array of integers $a$
  - variables $x$ and $y$
  - value of $y$ is value of $a$ at $x$-th position
  - in particular: $0 \leq x < \text{elements in } a$

- In Gecode
  - `element(*this, a, x, y);`
  - also for arrays of variables
Model with Element

IntArgs prices(4, 10, 15, 5, 12);
element(*this, prices, g, p);

- Just single propagator!
- Okay, if same integer occurs multiply in array
Propagating Element

- We insist on domain-consistency
  - bounds-consistency too weak

- For \( a[[x]] = y \) and store \( s \) propagate
  - if \( j \in s(y) \) then keep all \( k \) from \( s(x) \) with \( j = a[k] \)
  - if \( k \in s(x) \) then keep all \( j \) from \( s(y) \) with \( j = a[k] \)
  - remove all other values
Implementing Element...

- Fundamental requirement: new domains must be computed in order!
- Iterate over all elements $k \in s(x)$
  \[
  \{ a[k] \mid k \in s(x) \} \cap s(y)
  \]
- Iterate $k$ from 0 to $n-1:=\text{width of } a$
  construct new domain for $x$
  - if $a[k] \in s(y)$ then keep $k$
  - requires intersection and sorting
Problems…

- Array in element constraints can be very large
  - always iterate over entire array
  - always sort (or maintain sorted data structure)
  - always compute intersection

- We can do better!
Running Example

Consider $a[[x]] = y$ with

- $a = (4,5,9,7)$
- $s(x) = \{1,2,3\}$
- $s(y) = \{2\ldots8\}$

Propagation yields

- $s(x) = \{1,3\}$
- $s(y) = \{5,7\}$
Approach

- Construct data structure
  - contains pairs of \((i, a[i])\)
  - allows traversal for increasing \(i\)
  - allows traversal for increasing \(a[i]\)
  - allows removal of pairs (later)

- Data structure constructed initially
Datastructure Construction

- Iterate over all $i$ between 0 and 3
  - create node $(i, a[i])$
  - create links in order of creation ($x$-links)
Datastructure Construction

- Create links for $a[i]$ values in increasing order (y-links)
  - sort and create links
Datastructure Invariant

- Datastructure allows iteration
  - $i$ values in order: follow $x$-links
  - $a[i]$ values in order: follow $y$-links
Propagation

- Follow $x$-links and iterate values in $s(x)$
  - if value not in $s(x)$, remove node

- Follow $y$-links and iterate values in $s(y)$
  - if value not in $s(y)$, remove node

- Result: nodes with correct values remain for both $x$ and $y$
  - in increasing order!
Follow $x$-links...

- Store $s(x) = \{1,2,3\}$
- Remove node for 0
  - by relinking
  - constant time: doubly-linked lists
Follow $\gamma$-links…

- Store $s(y) = \{2, \ldots, 8\}$
- Remove node for 9
  - by relinking
  - constant time: doubly-linked lists
Read-off Variable Domains

- New store: \( s(x) = \{1,3\} \), \( s(y) = \{5,7\} \)
- By just following respective links
  - are sorted
  - are smaller than original domains
Incremental Propagation

- One option: destroy data structure
- Better: keep data structure for next propagator invocation
  - propagators with state
  - our model: propagator rewriting
- Incremental propagation
  - construction only initially
  - sorting only initially
  - traversing never for full number of array elements
Summary: Element

- Element important constraint for mapping variables to values
  - cost functions
  - arbitrary constraints defined extensionally

- Important for propagation
  - maintain clever data structure
  - make propagation incremental