Constraint Programming

Introduction, State of the Art & Trends

Christian Schulte
cschulte@kth.se

Department of Electronic, Computer and Software Systems
School of Information and Communication Technology
KTH – Royal Institute of Technology
Sweden
Talk Overview

- What is Constraint Programming?

  Sudoku is Constraint Programming

- ... more later
Sudoku

...is Constraint Programming!
Assign blank fields digits such that:
digits distinct per rows, columns, blocks
Sudoku

Assign blank fields digits such that: digits distinct per rows, columns, blocks
Assign blank fields digits such that: digits distinct per rows, columns, blocks.
Assign blank fields digits such that: digits distinct per rows, columns, **blocks**
Block Propagation

- No field in block can take digits 3, 6, 8
Block Propagation

- No field in block can take digits 3, 6, 8
  - propagate to other fields in block
- Rows and columns: likewise
Prune digits from fields such that:
digits distinct per rows, columns, blocks
Prune digits from fields such that:
digits distinct per rows, columns, blocks
Propagating digits from fields such that:
digits distinct per rows, columns, blocks
Propagate digits from fields such that:
digits distinct per rows, columns, blocks
Iterated Propagation

- Iterate propagation for rows, columns, blocks
- What if no assignment: search... later
# Sudoku is Constraint Programming

- **Variables**: fields
  - take **values**: digits
  - maintain set of possible values

- **Constraints**: distinct
  - relation among variables

- **Modelling**: variables, values, constraints

- **Solving**: propagation, search

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th>2</th>
<th></th>
<th>5</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td></td>
<td></td>
<td>7</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>9</td>
<td></td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>4</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td></td>
<td>1</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td></td>
<td></td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Constraint Programming

- Variable domains
  - finite domain integer, finite sets, multisets, intervals, ...

- Constraints
  - distinct, arithmetic, scheduling, graphs, ...

- Solving
  - propagation, branching, exploration, ...

- Modelling
  - variables, values, constraints, heuristics, symmetries, ...
Key ideas and principles
- constraint propagation
- search: branching and exploration

Why does constraint programming matter

State of the art and trends

Excursions
- constraint propagation revisited
- scheduling resources
- strong propagation
Key Ideas and Principles
Running Example: SMM

- Find distinct digits for letters, such that

\[
\begin{align*}
\text{SEND} & + \text{MORE} \\
\hline
\text{MONEY} & 
\end{align*}
\]
Constraint Model for SMM

- **Variables:**
  \( S, E, N, D, M, O, R, Y \in \{0, \ldots, 9\} \)

- **Constraints:**
  \[
  \text{distinct}(S, E, N, D, M, O, R, Y)
  \]
  \[
  1000 \times S + 100 \times E + 10 \times N + D
  \]
  \[
  + 1000 \times M + 100 \times O + 10 \times R + E
  \]
  \[
  = 10000 \times M + 1000 \times O + 100 \times N + 10 \times E + Y
  \]
  \[
  S \neq 0 \quad M \neq 0
  \]
Solving SMM

- Find values for variables

such that

all constraints satisfied
Finding a Solution

- Compute with possible values
  - rather than enumerating assignments

- Prune inconsistent values
  - constraint propagation

- Search
  - branch: define search tree
  - explore: explore search tree for solution
Constraint Propagation
Important Concepts

- Constraint store
- Propagator
- Constraint propagation
Constraint Store

\[ x \in \{3,4,5\} \quad y \in \{3,4,5\} \]

- Maps variables to possible values
Constraint Store

- Maps variables to possible values
- Others: finite sets, intervals, trees, ...

finite domain constraints

\(x \in \{3,4,5\} \quad y \in \{3,4,5\}\)
Propagators

- Implement constraints

\[ \text{distinct}(x_1, \ldots, x_n) \]

\[ x + 2xy = z \]
Propagators

- \( x \geq y \) \quad \text{and} \quad y > 3

- \( x \in \{3, 4, 5\} \quad \text{and} \quad y \in \{3, 4, 5\} \)

- Amplify store by constraint propagation
Propagators

- Amplify store by constraint propagation
Propagators

- Amplify store by constraint propagation

\[ x \geq y \quad y > 3 \]

\[ x \in \{3,4,5\} \quad y \in \{4,5\} \]
Propagators

- Amplify store by constraint propagation
Propagators

- Amplify store by constraint propagation

\( x \geq y \quad y > 3 \)

\( x \in \{4,5\} \quad y \in \{4,5\} \)
Propagators

- Amplify store by constraint propagation
- Disappear when done (subsumed, entailed)
  - no more propagation possible

\[ x \geq y \quad y > 3 \]
\[ x \in \{4, 5\} \quad y \in \{4, 5\} \]
Propagators

- Amplify store by constraint propagation
- Disappear when done (subsumed, entailed)
  - no more propagation possible

\[ x \geq y \]
\[ x \in \{4, 5\} \quad y \in \{4, 5\} \]
Propagation for SMM

- Results in store

\[
S \in \{9\} \quad E \in \{4, \ldots, 7\} \quad N \in \{5, \ldots, 8\} \quad D \in \{2, \ldots, 8\} \\
M \in \{1\} \quad O \in \{0\} \quad R \in \{2, \ldots, 8\} \quad Y \in \{2, \ldots, 8\}
\]

- Propagation **alone** not sufficient!
  - create simpler sub-problems
  - branching
Important Concepts

- Branching
- Exploration
- Branching heuristics
- Best solution search
Search: Branching

- Create subproblems with additional information
  - enable further constraint propagation
Example Branching Strategy

- Pick variable $x$ with at least two values
- Pick value $n$ from domain of $x$
- Branch with
  \[ x = n \quad \text{and} \quad x \neq n \]

- Part of model
Search: Exploration

- Iterate propagation and branching
- Orthogonal: branching ⇔ exploration
- Nodes:
  - Unsolved
  - Failed
  - Succeeded
SMM: Solution

\[
\begin{align*}
\text{SEND} & \quad + \quad \text{MORE} \\
\phantom{+} & \quad = \quad \text{MONEY} \\
9567 & \quad + \quad 1085 \\
\hline
10652 & \quad = \quad \text{MONEY}
\end{align*}
\]
Heuristics for Branching

- Which variable
  - least possible values (first-fail)
  - application dependent heuristic

- Which value
  - minimum, median, maximum
    \[ x = m \quad \text{or} \quad x \neq m \]
  - split with median \( m \)
    \[ x < m \quad \text{or} \quad x \geq m \]

- Problem specific
SMM: Solution With First-fail

\[
\begin{align*}
\text{SEND} & \quad + \quad \text{MORE} \\
= & \quad \text{MONEY} \\
= & \quad 9567 \\
+ & \quad 1085 \\
= & \quad 10652
\end{align*}
\]
Send Most Money (SMM++)

- Find distinct digits for letters, such that

\[
\begin{align*}
\text{SEND} & \quad + \quad \text{MOST} \\
\hline
\text{MONEY} & =
\end{align*}
\]

and \text{MONEY} maximal
Best Solution Search

- **Naïve approach:**
  - compute all solutions
  - choose best

- **Branch-and-bound approach:**
  - compute first solution
  - add “betterness” constraint to open nodes
  - next solution will be “better”
  - prunes search space
Branch-and-bound Search

Find first solution
Branch-and-bound Search

- Explore with additional constraint
Branch-and-bound Search

- Explore with additional constraint
Branch-and-bound Search

 Guarantees better solutions
Branch-and-bound Search

Guarantees better solutions
Branch-and-bound Search

Last solution best
Branch-and-bound Search

Proof of optimality
Modelling SMM++

- Constraints and branching as before
- Order among solutions with constraints
  - so-far-best solution \(S, E, N, D, M, O, T, Y\)
  - current node \(S, E, N, D, M, O, T, Y\)
  - constraint added
    \[10000 \times M + 1000 \times O + 100 \times N + 10 \times E + Y < 10000 \times M + 1000 \times O + 100 \times N + 10 \times E + Y\]
SMM++: Branch-and-bound

\[
\begin{align*}
\text{SEND} & + \text{MOST} \\
& = \text{MONEY} \\
& = 9782 + 1094 \\
& = 10876
\end{align*}
\]
SMM++: All Solution Search

SEND
+ MOST
= MONEY

\[ 9782 + 1094 = 10876 \]
Summary: Key Ideas and Principles

- **Modelling**
  - variables with domain
  - constraints to state relations
  - branching strategy
  - solution ordering

- **Solving**
  - constraint propagation
  - constraint branching
  - search tree exploration

- **Applications**

- **Principles**
Excursion
Constraint Propagation
Revisited
Constraint Propagation

- Variables (as members of store)
  - feature variable domain (here: finite set of integers)

- Propagators
  - implement constraints

- Propagation loop
  - execute propagators until simultaneous fixpoint
Propagator

- Propagator $p$ is procedure
  - implements constraint $\text{con}(p)$
  - its semantics (set of tuples)
  - computes on set of variables $\text{var}(p)$

- Execution of propagator $p$
  - narrows domains of variables in $\text{var}(p)$
  - signals failure
Propagators Are Intensional

- Propagators implement narrowing
  - also: filtering, propagation, domain reduction

- No extensional representation of con(\(\rho\))
  - impractical in most cases (space)

- Extensional representation of constraint
  - can be provided by special propagator
  - often: “element” constraint, “relation” constraint, …
Propagator Properties

- Propagator $p$ is
  - correct: no solution of $\text{con}(p)$ is removed
  - assignment complete: failure at latest for assignments
    - compatibility with search

- Propagator $p$ is
  - contracting: variable domains are narrowed
  - monotonic: application to smaller domains will result in smaller domains than application to larger domains
Propagation Loop

- **Largest simultaneous fixpoint of propagators**
  - fixpoint: propagators cannot narrow any further
  - largest: no solutions lost

- **Guaranteed**
  - termination: domains finite
  - largest fixpoint: propagators contracting

Detailed study with proofs: [Apt 00]
Fix and Runnable Propagators

- Propagator is either
  - fix: has reached fixpoint
  - runnable: not known to have reached fixpoint

- Propagation loop maintains propagator sets
  - all propagators \( Prop \)
  - runnable propagators \( Run \)
  - initially \( Run := Prop \)
Sketch of Propagation Loop

\[
\textbf{while} (\text{Run} \neq \emptyset) \{ \\
\text{pick and remove } p \text{ from } \text{Run}; \\
\text{execute } p; \\
\text{ModVar} := \{ x | x \text{ modified by } p \}; \\
\text{DepProp} := \{ q | x \in \text{var}(q), x \in \text{ModVar} \}; \\
\text{Run} := \text{join}(\text{DepProp}, \text{Run}); \\
\} 
\]
Sketch of Propagation Loop

**while** \((Run \neq \emptyset)\) {
    pick and remove \(p\) from \(Run\);
    execute \(p\);
    \(ModVar := \{ x \mid x \text{ modified by } p \}\);
    \(DepProp := \{ q \mid x \in \text{var}(q), x \in ModVar \}\);
    \(Run := \text{join}(DepProp, Run)\);
}

Loop invariant: \(p\) is fix \(\iff p \in (Prop-Run)\)
Sketch of Propagation Loop

\[
\text{while } (Run \neq \emptyset) \{ \\
\quad \text{pick and remove } p \text{ from } Run; \\
\quad \text{execute } p; \\
\quad \text{ModVar} := \{ x \mid x \text{ modified by } p \}; \\
\quad \text{DepProp} := \{ q \mid x \in \text{var}(q), x \in \text{ModVar} \}; \\
\quad \text{Run} := \text{join}(\text{DepProp}, \text{Run}); \\
\}\n\]

Termination \((Run=\emptyset)\): \(p \text{ is fix } \iff p \in Prop\)
Sketch of Propagation Loop

\[\textbf{while} \ (\textit{Run} \neq \emptyset) \ {\textbf{\{}} \]

\begin{align*}
\text{pick and remove } p \text{ from } \textit{Run}; \\
\text{execute } p; \\
\textit{ModVar} := \{x \mid x \text{ modified by } p\}; \\
\textit{DepProp} := \{q \mid x \in \text{var}(q), \ x \in \textit{ModVar}\}; \\
\textit{Run} := \text{join}(\textit{DepProp}, \textit{Run}); \\
\textbf{\}}
\end{align*}

\textbf{Ignored: failure (signaled by } p)\]
Implementing *ModVar* and *DepProp*

- **Variable-centered approach**
  - each variable $x$ knows dependent propagators
  - typically organized as list (*suspension list*)
  - propagator $p$ included in list of $x \iff x \in \text{var}(p)$

- **Upon propagator creation**
  - propagator subscribes to its variables
  - becomes runnable
Propagators $\Rightarrow$ Variables

- **Propagators** know their variables
  - to perform domain modifications
  - passed as parameters to propagator creation
Variables $\Rightarrow$ Propagators

- **Variables** know dependent **propagators**
  - to perform efficient computation of dependent propagators
Modifying a Variable

- Traverse suspension list
  - add propagators to *Run*

- Optimization
  - mark runnable propagators
  - that is: propagators already in *Run*

- Multiple variable modification by propagator
  - explicitly maintain *ModVar* (as in model)
  - only after propagator execution: process *ModVar*
  - suspension list traversed only once per variable
Idempotent Propagators

- Idempotent propagator
  - always computes fixpoint

- Propagation loop perspective
  - no need to include in Run
  - more efficient: saves one invocation of propagator

- Propagator perspective
  - must compute fixpoint itself
  - more efficient: specific method for computing fixpoint
  - might be more challenging
Propagator Entailment

- Propagator will never contribute anything
  - fixpoint property preserved by narrowing

- Delete propagator, if entailment detected
  - remove from suspension-list, or
  - mark as dead, delegate removal to garbage collection
Summary: Constraint Propagation Revisited

- **Variables**
  - domain, suspension list

- **Propagators**
  - intensional, correct, contracting, monotone, ...
  - know variables for narrowing

- **Propagation loop**
  - computes largest simultaneous fixpoint
Why Does Constraint Programming Matter
Widely Applicable

- Timetabling
- Scheduling
- Crew rostering
- Resource allocation
- Workflow planning and optimization
- Gate allocation at airports
- Sports-event scheduling
- Railroad: track allocation, train allocation, schedules
- Automatic composition of music
- Genome sequencing
- Frequency allocation
- …
Draws on Variety of Techniques

- Artificial intelligence
  - basic idea, search, ...
- Operations research
  - scheduling, flow, ...
- Algorithms
  - graphs, matching, networks, ...
- Programming languages
  - programmability, extensionability, ...
Essential Aspect

- Compositional middleware for combining
  - smart algorithmic
  - problem substructures

  components (propagators)
  - scheduling
  - graphs
  - flows
  - ...

  plus
  - essential extra constraints
Significance

- Constraint programming identified as a strategic direction in computer science research
  [ACM Computing Surveys, December 1996]
Excursion
Scheduling Resources

- Modelling
- Propagation
- Strong propagation
Scheduling Resources: Problem

- **Tasks**
  - duration
  - resource

- **Precedence constraints**
  - determine order among two tasks

- **Resource constraints**
  - at most one task per resource
    [disjunctive, non-preemptive scheduling]
Scheduling: Bridge Example

Infamous: additional side constraints!
Scheduling: Solution

- Start time for each task
- All constraints satisfied
- Earliest completion time
  - minimal make-span
Scheduling: Model

- Variable for start-time of task $a$
  \[
  \text{start}(a)
  \]
- Precedence constraint: $a$ before $b$
  \[
  \text{start}(a) + \text{dur}(a) \leq \text{start}(b)
  \]
Propagating Precedence

\[ a \text{ before } b \]

\[ \text{start}(a) \in \{0, \ldots, 7\} \]
\[ \text{start}(b) \in \{0, \ldots, 5\} \]
Propagating Precedence

\[ a \text{ before } b \]

\[ \begin{align*}
\text{start}(a) &\in \{0, \ldots, 7\} \\
\text{start}(b) &\in \{0, \ldots, 5\}
\end{align*} \]

\[ \begin{align*}
\text{start}(a) &\in \{0, \ldots, 2\} \\
\text{start}(b) &\in \{3, \ldots, 5\}
\end{align*} \]
Scheduling: Model

- Variable for start-time of task \( a \)
  \[ \text{start}(a) \]

- Precedence constraint: \( a \) before \( b \)
  \[ \text{start}(a) + \text{dur}(a) \leq \text{start}(b) \]

- Resource constraint:
  \( a \) before \( b \)
  or
  \( b \) before \( a \)
Scheduling: Model

- Variable for start-time of task $a$
  \[ \text{start}(a) \]
- Precedence constraint: $a$ before $b$
  \[ \text{start}(a) + \text{dur}(a) \leq \text{start}(b) \]
- Resource constraint:
  \[ \text{start}(a) + \text{dur}(a) \leq \text{start}(b) \]
  or
  \[ b \text{ before } a \]
Scheduling: Model

- Variable for start-time of task $a$
  \[
  \text{start}(a)
  \]
- Precedence constraint: $a$ before $b$
  \[
  \text{start}(a) + \text{dur}(a) \leq \text{start}(b)
  \]
- Resource constraint:
  \[
  \text{start}(a) + \text{dur}(a) \leq \text{start}(b)
  \]
  or
  \[
  \text{start}(b) + \text{dur}(b) \leq \text{start}(a)
  \]
Reified Constraints

- Use control variable $b \in \{0,1\}$
  
  $c \leftrightarrow b=1$

- Propagate
  
  - $c$ holds $\Rightarrow$ propagate $b=1$
  - $\neg c$ holds $\Rightarrow$ propagate $b=0$
  - $b=1$ holds $\Rightarrow$ propagate $c$
  - $b=0$ holds $\Rightarrow$ propagate $\neg c$
Reified Constraints

- Use control variable $b \in \{0,1\}$
  \[ c \leftrightarrow b=1 \]

- Propagate
  - $c$ holds \[ \Rightarrow \text{propagate } b=1 \]
  - $\neg c$ holds \[ \Rightarrow \text{propagate } b=0 \]
  - $b=1$ holds \[ \Rightarrow \text{propagate } c \]
  - $b=0$ holds \[ \Rightarrow \text{propagate } \neg c \]

\[ \text{not easy!} \]
Reification for Disjunction

- Reify each precedence
  \[\text{start}(a) + \text{dur}(a) \leq \text{start}(b)\] \iff \(b_0 = 1\)

and

\[\text{start}(b) + \text{dur}(b) \leq \text{start}(a)\] \iff \(b_1 = 1\)

- Model disjunction
  \[b_0 + b_1 \geq 1\]
Model Is Too Naive

- **Local view**
  - individual task pairs
  - $O(n^2)$ propagators for $n$ tasks

- **Global view ("global" constraints)**
  - all tasks on resource
  - single propagator
  - smarter algorithms possible
Example: Edge Finding

- Find ordering among tasks ("edges")
- For each subset of tasks \( \{a\} \cup B \)
  - assume: \( a \) before \( B \)
    - deduce information for \( a \) and \( B \)
  - assume: \( B \) before \( a \)
    - deduce information for \( a \) and \( B \)
  - join computed information
  - can be done in \( O(n^2) \)
Summary

- Modelling
  - easy but not always efficient
  - constraint combinators (reification)
  - global constraints
  - smart heuristics

- More on constraint-based scheduling
Excursion
Strong Propagation
SMM: Strong Propagation

\[
\begin{align*}
\text{SEND} & \quad + \quad \text{MORE} \\
& \quad = \quad \text{MONEY} \\
& \quad = \quad 9567 \\
& \quad + \quad 1085 \\
& \quad = \quad 10652
\end{align*}
\]
Example: Distinct Propagator

- **Infeasible: decomposition**
  - $O(n^2)$ disequality propagators

- **Naive distinct propagator**
  - wait until variable becomes assigned
  - remove value from all other variables

- **Strong distinct propagator**
  - only keep values appearing in a solution to constraint
  - essential for many problems
Distinct Propagator: Hall Sets

- Direct approach: Hall sets
  - Van Beek, Quimper, et. al. [CP 2004]

- Set \( \{x_1, ..., x_n\} \) of variables Hall set, iff set of values \( s(x_1) \cup ... \cup s(x_n) \) has cardinality \( n \)

- Pruning
  - find Hall set \( H \)
  - prune values in \( H \) from all other variables
Strong Distinct Propagator

- Can be propagated efficiently
  - \(O(n^{2.5})\) is efficient

- Uses graph algorithms
  - insight on problem structure
  - relation between solutions of constraint and properties of graph
Régin's Approach

- Construct a variable-value graph
  - bipartite graph: variable nodes $\rightarrow$ value nodes
- Characterize solutions in graph
  - maximal matchings
- Use matching theory
  - one matching can describe all matchings
- Remove edges not representing solutions
Variable Value Graph

\[ s(x_0) = \{0, 1\} \]
\[ s(x_1) = \{1, 2\} \]
\[ s(x_2) = \{0, 2\} \]
\[ s(x_3) = \{1, 3\} \]
\[ s(x_4) = \{2, 3, 4, 5\} \]
\[ s(x_5) = \{5, 6\} \]

Diagram:
- \( x_0 \) connects to 0 and 1
- \( x_1 \) connects to 1 and 2
- \( x_2 \) connects to 0 and 2
- \( x_3 \) connects to 1 and 3
- \( x_4 \) connects to 2, 3, 4, and 5
- \( x_5 \) connects to 5 and 6
Maximal Matching Are Solutions

\[ a(x_0) = 0 \]
\[ a(x_1) = 1 \]
\[ a(x_2) = 2 \]
\[ a(x_3) = 3 \]
\[ a(x_4) = 4 \]
\[ a(x_5) = 6 \]
Matching Theory

- Edge $e$ belongs to some matching $\leftrightarrow$ for some arbitrary matching $M$:
  - either: $e$ belongs to even alternating path starting at free node
  - or: $e$ belongs to even alternating cycle

- [C. Berge, 1970] See Régin's paper
Oriented Graph: Alternation
Alternating Paths…

- Only free node: 6
  - mark 6 $\rightarrow$ $x_5$
  - mark $x_5$ $\rightarrow$ 5
  - mark 5 $\rightarrow$ $x_4$
  - mark $x_4$ $\rightarrow$ 4

- Intuition: edges can be permuted
Alternating Cycles…

- Nodes in SCC
  \( x_0, x_1, x_2, 0, 1, 2 \)
- Mark joining edges
- Intuition: variables take all values from SCC
All Marked Edges
Edges Removed

- **Remove**
  - \(1 \rightarrow x_3\)
  - \(2 \rightarrow x_4\)
  - \(3 \rightarrow x_4\)

- **Keep**
  - \(x_3 \rightarrow 3\)
  - matched!

- **Edge removal**
  - value removal
Characterising Strength: Consistency

- **Domain-consistent propagator for constraint**
  - every value appears in at least one solution of constraint
  - strongest possible propagation
  - Régis's method is domain-consistent
  - also known as: generalized arc consistency, ...

- ** Bounds-consistent propagator for constraint**
  - extremal values appears in solution of convex relaxation
  - depends on relaxation: integer versus real
  - weaker but cheaper yet relevant
  - confusion about variants...
Global Constraints

- Reasons for globality: decomposition...
  - semantic: ...not possible
  - operational: ...less propagation
  - algorithmic: ...less efficiency

- Plethora available
  - scheduling, sequencing, cardinality, sorting, circuits, ...
  - systematic catalogue with hundreds available...
  - difficult to pick the right one (consistency versus efficiency, etc)
Trends and State of the Art
Trends and State of The Art

- Focus here
  - constraints for combinatorial problems
  - ignoring
    - programming languages, graphics, databases, tractability, complexity, ...

- Up-to-date overview
  Handbook of Constraint Programming
Modelling

- Symmetry breaking
- Implied constraints
- Variable domains
- Soft constraints
- Modelling languages
- ...

Christian Schulte, ICT, KTH
Symmetry Breaking

- Absolutely essential
  - just search for single solution, ignore symmetric solutions
  - drastically prunes search space
  - without, most problems can not be solved

- Key questions
  - how to find symmetries automatically?
  - class of symmetries: value, variable symmetries?
  - how to break them (rule out symmetric solutions)_DISTILLATION_BEGIN?
  - how many to break (all typically to expensive)?
  - break them statically or dynamically?
  - break them during search?
Implied Constraints

- Absolutely essential
  - find constraints that are semantically implied
  - yet provide essential propagation

- Key questions
  - how to find them?
  - manual versus automatic?
  - how to propagate them?
Variable Domains

- Finite sets, multisets, intervals, ...
- Often help to avoid symmetries (sets)
- Typically require approximation
  - full set representation: exponential time and space
  - bounds approximation: describe by glb and lub

Key questions
- total ordering for symmetry breaking?
- efficient representations?
- efficient and strong propagators for global constraints?
Soft Constraints

- Important to capture inconsistent models
  - as they tend to be in practice
- Devise new framework
  - generalize propagation to cater for softness
- Remain in same framework
  - propagators that propagate according to degree of violation
- ...
Modelling Languages

- Fundamental difference to LP and SAT
  - language has structure (global constraints)
  - different solvers support different constraints

- In its infancy

- Key questions:
  - what level of abstraction?
  - solving approach independent: LP, SAT, CP, ...?
  - how to map to different systems?
Solving

- Automatic solving ("black box" solvers)
- Constraint-based local search
- Hybrid approaches
- Constraint programming systems
- Global constraints
- ...
Automatic Solving

- Modelling is very difficult for CP
  - requires lots of knowledge and tinkering
  - very different from SAT

- How to automatize
  - restart search?
  - automatic symmetric breaking?
  - new idea, promising first ideas and approaches?
  - to which extent possible?
Constraint-based Local Search

- Local search
  - operate on assignments not necessarily solutions
  - find "good" assignments
- Use constraints as abstractions to model and solve with local search
- Derive implementations automatically from constraints
- Hybrid approaches?
- Very promising
  - check out Comet: www.comet-online.org
Hybrid Approaches

- Operations research methods
- Key issue: CP poor for optimization
- Key questions
  - relaxations to obtain bounds?
  - column generation?
  - Benders decomposition?
  - cuts?
- Extremely important for practical problems
Global Constraints

■ Ever more! Ever more?

■ Key questions
  ■ what are the essential primitive ones?
  ■ how to characterize them?
  ■ how to automatically get an implementation?
Constraint Programming Systems

- Essential for initial and continuing success
- Two approaches
  - library-based: ILOG Solver, Koalog, Choco, Gecode, ...
  - language-based: SICStus Prolog, Eclipse, Oz, ...
- Key questions
  - parallelism
  - efficiency
  - robustness
  - automatic
  - coverage
Constraint Programming

- Powerful approach for modelling and solving combinatorial problems
- Key aspect: middleware for
  - powerful algorithmic components
  - essential extra constraints

- Key issues: modelling, propagation, search

- Widely used but modelling is challenging